Ada Programming

Wikibooks.org

March 22, 2013

Contents

1 Basic Ada

1.1 "Hello, world!" programs

1.1.1 "Hello, world!"

A common example of a language's syntax[1] is the Hello world program[2]. Here is a straightforward Ada Implementation:

File: hello_world_1.adb

```
with Ada.Text_IO

procedure Hello is
begin
   Ada.Text_IO.Put_Line("Hello, world!");
end Hello;
```

The **with** statement adds the package `Ada.Text_IO` to the program. This package comes with every Ada compiler and contains all functionality needed for textual Input/Output. The **with** statement makes the declarations of `Ada.Text_IO` available to procedure `Hello`. This includes the types declared in `Ada.Text_IO`, the subprograms of `Ada.Text_IO` and everything else that is declared in `Ada.Text_IO` for public use. In Ada, packages can be used as toolboxes. `Text_IO` provides a collection of tools for textual input and output in one easy-to-access module. Here is a partial glimpse at package Ada.Text_IO

```
package Ada.Text_IO is

   type File_Type is limited private;

   --morestuff

   procedure Open(File : in out File_Type;
                  Mode : File_Mode;
                  Name : String;
                  Form : String := "");

   --morestuff

   procedure Put_Line (Item : String);

   --morestuff
```

1 http://en.wikipedia.org/wiki/Syntax_%28programming_languages%29
2 http://en.wikipedia.org/wiki/Hello_world_program

end Ada.Text_IO;

Next in the program we declare a main procedure. An Ada main procedure does not need to be called "main". Any simple name is fine so here it is *Hello*. Compilers might allow procedures or functions to be used as main subprograms. [3]

The call on `Ada.Text_IO.Put_Line` writes the text "Hello World" to the current output file.

A **with** clause makes the content of a package *visible by selection*: we need to prefix the procedure name `Put_Line` from the `Text_IO` package with its full package name `Ada.Text_IO`. If you need procedures from a package more often some form of shortcut is needed. There are two options open:

1.1.2 "Hello, world!" with renames

By renaming a package it is possible to give a shorter alias to any package name.[4] This reduces the typing involved while still keeping some of the readability.

File: hello_world_2.adb

```
with Ada.Text_IO

procedure Hello is
   package IO renames Ada.Text_IO;
begin
   IO.Put_Line("Hello, world!");
   IO.New_Line;
   IO.Put_Line("I am an Ada program with package rename.");
end Hello;
```

1.1.3 "Hello, world!" with use

The **use** clause makes all the content of a package directly visible. It allows even less typing but removes some of the readability. One suggested "rule of thumb": **use** for the most used package and **renames** for all other packages. You might have another rule (for example, always **use** Ada.Text_IO. never **use** anything else).

3 Main subprograms may even have parameters; it is implementation-defined what kinds of subprograms can be used as main subprograms. The reference manual explains the details in 10.2 LRM 10.2(29) ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-10-2.html} : "..., an implementation is required to support all main subprograms that are public parameterless library procedures." *Library* means not nested in another subprogram, for example, and other things that needn't concern us now.

4 **renames** can also be used for procedures, functions, variables, array elements. It can not be used for types - a type rename can be accomplished with **subtype**.

File: hello_world_3.adb

```
with Ada.Text_IOuse Ada.Text_IO;

procedure Hello is

begin
   Put_Line("Hello, world!");
   New_Line;
   Put_Line("I am an Ada program with package use.");
end Hello;
```

use can be used for packages and in the form of **use type** for types. **use type** makes only the operators[5] of the given type directly visible but not any other operations on the type.

1.2 Compiling the "Hello, world!" program

For information on how to build the "Hello, world!" program on various compilers, see the Building[6] chapter.

1.2.1 FAQ: Why is "Hello, world!" so big?

Ada beginners frequently ask how it can be that such a simple program as "Hello, world!" results in such a large executable. The reason has nothing to do with Ada but can usually be found in the compiler and linker options used — or better, not used.

Standard behavior for Ada compilers — or good compilers in general — is not to create the best code possible but to be optimized for ease of use. This is done to ensure a system that works "out of the box" and thus does not frighten away potential new users with unneeded complexity.

The GNAT project files, which you can download[7] alongside the example programs, use better tuned compiler, binder and linker options. If you use those your "Hello, world!" will be a lot smaller:

```
 32K ./Linux-i686-Debug/hello_world_1
8.0K ./Linux-i686-Release/hello_world_1
 36K ./Linux-x86_64-Debug/hello_world_1
 12K ./Linux-x86_64-Release/hello_world_1
1.1M ./Windows_NT-i686-Debug/hello_world_1.exe
 16K ./Windows_NT-i686-Release/hello_world_1.exe
 32K ./VMS-AXP-Debug/hello_world_1.exe
 12K ./VMS-AXP-Release/hello_world_1.exe
```

5 Chapter 37 on page 301
6 Chapter 3 on page 29
7 https://sourceforge.net/project/showfiles.php?group_id=124904

For comparison the sizes for a plain **gnat make** compile:

```
497K hello_world_1 (Linux i686)
500K hello_world_1 (Linux x86_64)
1.5M hello_world_1.exe (Windows_NT i686)
589K hello_world_1.exe (VMS AXP)
```

Worth mentioning is that hello_world (Ada,C,C++) compiled with GNAT/MSVC 7.1/GCC(C) all produces executables with approximately the same size given comparable optimisation and linker methods.

1.3 Things to look out for

It will help to be prepared to spot a number of significant features of Ada that are important for learning its syntax and semantics.

1.3.1 Comb Format

There is a *comb format* in all the control structures and module structures. See the following examples for the *comb format*. You don't have to understand what the examples do yet - just look for the similarities in layout.

```
if Boolean expression then
   statements
elsif Boolean expression then
   statements
else
   statements
end if;
```

```
while Boolean expression loop
   statements
end loop;
```

```
for variable in range loop
   statements
end loop;
```

```
declare
   declarations
begin
   statements
exception
   handlers
end;
```

```
procedure P (parameters : in out type) is
   declarations
begin
   statements
```

```
exception
   handlers
end P;
```

```
function F (parameters : in type) return type is
   declarations
begin
   statements
exception
   handlers
end F;
```

```
package P is
   declarations
private
   declarations
end P;
```

```
generic
   declarations
package P is
   declarations
private
   declarations
end P;
```

```
generic
   declarations
procedure P (parameters : in out type);
```

Note that semicolons consistently terminate statements and declarations; the empty line (or a semicolon alone) is not a valid statement: the null statement is

```
null;
```

1.3.2 Type and subtype

There is an important distinction between **type** and **subtype**: a type is given by a set of values and their operations. A subtype is given by a type, and a *constraint* that limits the set of values. Values are always of a type. Objects (constants and variables) are of a subtype. This generalizes, clarifies and systematizes a relationship, e.g. between *Integer* and 1..100, that is handled *ad hoc* in the semantics of Pascal[8].

1.3.3 Constrained types and unconstrained types

There is an important distinction between *constrained* types and *unconstrained* types. An unconstrained type has one or more free parameters that affect its size or shape. A constrained type fixes the values of these parameters and so determines its size and shape.

8 http://en.wikipedia.org/wiki/Pascal%20programming%20language

Loosely speaking, objects must be of a constrained type, but formal parameters may be of an unconstrained type (they adopt the constraint of any corresponding actual parameter). This solves the problem of array parameters in Pascal (among other things).

1.3.4 Dynamic types

Where values in Pascal[9] or C[10] must be static (e.g. the subscript bounds of an array) they may be dynamic in Ada. However, static expressions are required in certain cases where dynamic evaluation would not permit a reasonable implementation (e.g. in setting the number of digits of precision of a floating point type).

1.3.5 Separation of concerns

Ada consistently supports a separation of interface and mechanism. You can see this in the format of a package[11], which separates its declaration from its body; and in the concept of a private type, whose representation in terms of Ada data structures is inaccessible outside the scope containing its definition.

1.4 Where to ask for help

Most Ada experts lurk on the Usenet newsgroups[12] *comp.lang.ada* (English) and *fr.comp.lang.ada* (French); they are accessible either with a newsreader[13] or through one of the many web interfaces. This is the place for all questions related to Ada.

People on these newsgroups are willing to help but will *not* do students' homework for them; they will not post complete answers to assignments. Instead, they will provide guidance for students to find their own answers.

For more online resources, see the External links[14] section in this wikibook's introduction.

1.5 Notes

9 http://en.wikipedia.org/wiki/Pascal%20programming%20language
10 http://en.wikipedia.org/wiki/C%20programming%20language
11 Chapter 17 on page 133
12 http://en.wikipedia.org/wiki/Newsgroup
13 http://en.wikipedia.org/wiki/News_client
14 http://en.wikibooks.org/wiki/Ada%20Programming%23External%20links

2 Installing

Ada compilers[1] are available from several vendors, on a variety of host and target platforms. The Ada Resource Association[2] maintains a list of available compilers[3].

Below is an alphabetical list of available compilers with additional comments.

2.1 AdaMagic from SofCheck

SofCheck[4] produces an Ada 95 front-end that can be plugged into a code generating back-end to produce a full compiler. This front-end is offered for licensing to compiler vendors.

Based on this front-end, SofCheck offers:

- AdaMagic, an Ada-to-C translator
- AppletMagic, an Ada-to-Java[5] bytecode compiler

Commercial; proprietary.

2.2 AdaMULTI from Green Hills Software

Green Hills Software sells development environments for multiple languages and multiple targets (including DSP[6]s), primarily to embedded software developers.

Languages supported	Ada 83, Ada 95, C, C++, Fortran
License for the run-time library	proprietary
Native platforms	GNU/Linux on i386, Microsoft Windows on i386, and Solaris on SPARC
Cross platforms	INTEGRITY, INTEGRITY-178B and velOSity from Green Hills; VxWorks from Wind River; several bare board targets. Safety-critical GMART and GSTART run-time libraries certified to DO-178B level A.
Available from	`http://www.ghs.com/`

1 http://en.wikipedia.org/wiki/Compiler
2 http://www.adaic.com
3 http://www.adaic.com/compilers/comp-tool.html
4 http://www.sofcheck.com/
5 http://en.wikibooks.org/wiki/Ada_Programming%2FPlatform%2FVM%2FJava
6 http://en.wikipedia.org/wiki/Digital%20signal%20processor

Support	Commercial
Add-ons included	IDE, debugger, TimeMachine, integration with various version control systems, source browsers, other utilities

GHS claims to make great efforts to ensure that their compilers produce the most efficient code and often cites the EEMBC[7] benchmark results as evidence, since many of the results published by chip manufacturers use GHS compilers to show their silicon in the best light, although these benchmarks are not Ada specific.

GHS has no publicly announced plans to support the new Ada standard published in 2007 but they do continue to actively market and develop their existing Ada products.

2.3 DEC Ada from HP

DEC Ada is an Ada 83 compiler for OpenVMS[8]. While "DEC Ada" is probably the name most users know, the compiler is now called " HP Ada[9]". It had previously been known also by names of "VAX Ada" and "Compaq Ada".

- Ada for OpenVMS Alpha Installation Guide[10] (PDF)
- Ada for OpenVMS VAX Installation Guide[11] (PDF)

2.4 GNAT, the GNU Ada Compiler from AdaCore and the Free Software Foundation

GNAT[12] is the free GNU Ada compiler, which is part of the GNU Compiler Collection[13]. It is the only Ada compiler that supports all of the optional annexes of the language standard. The original authors formed the company AdaCore[14] to offer professional support, consulting, training and custom development services. It is thus possible to obtain GNAT from many different sources, detailed below.

GNAT is always licensed under the terms of the GNU General Public License[15].

However, the run-time library uses either the GPL[16], or the GNAT Modified GPL[17], depending on where you obtain it.

7 http://www.eembc.com
8 http://en.wikipedia.org/wiki/OpenVMS
9 http://h71000.www7.hp.com/commercial/ada/ada_index.html
10 http://h71000.www7.hp.com/commercial/ada/ada_avms_ig.pdf
11 http://h71000.www7.hp.com/commercial/ada/ada_vvms_ig.pdf
12 http://en.wikipedia.org/wiki/GNAT
13 http://en.wikipedia.org/wiki/GNU_Compiler_Collection
14 http://www.adacore.com
15 http://en.wikipedia.org/wiki/GNU%20General%20Public%20License
16 http://en.wikipedia.org/wiki/GNU%20General%20Public%20License
17 http://en.wikipedia.org/wiki/GNAT%20Modified%20General%20Public%20License

Several optional add-ons are available from various places:

- ASIS, the Ada Semantic Interface Specification[18], is a library that allows Ada programs to examine and manipulate other Ada programs.
- FLORIST[19] is a library that provides a POSIX programming interface to the operating system.
- GDB, the GNU Debugger, with Ada extensions.
- GLADE implements Annex E, the Distributed Systems Annex. With it, one can write distributed programs in Ada, where partitions of the program running on different computers communicate over the network with one another and with shared objects.
- GPS, the GNAT Programming Studio, is a full-featured integrated development environment, written in Ada. It allows you to code in Ada, C and C++.

Many Free Software libraries are also available.

2.4.1 GNAT GPL Edition

This is a source and binary release from AdaCore, intended for use by Free Software developers only. If you want to distribute your binary programs linked with the GPL run-time library, then you must do so under terms compatible with the GNU General Public License.

As of GNAT GPL Edition 2011:

Languages supported	Ada 83, Ada 95, Ada 2005, Ada 2012, C, C++
License for the run-time library	pure GPL
Native platforms	GNU/Linux on i386 and x86_64; Microsoft Windows on i386; Microsoft .NET on i386; Mac OS X (Darwin, x86_64); Solaris on SPARC.
Cross platforms	AVR, hosted on Windows; Java VM, hosted on Windows; Mindstorms NXT, hosted on Windows
Compiler back-end	GCC 4.5.3
Available from	`http://libre.adacore.com/` (requires free registration)
Support	None
Add-ons included	GDB, GNATbench (Eclipse plug-in), GPS in source and binary form; many more in source-only form.

2.4.2 GNAT Modified GPL releases

With these releases of GNAT, you can distribute your programs in binary form under licensing terms of your own choosing; you are not bound by the GPL.

18 `http://en.wikipedia.org/wiki/Ada%20Semantic%20Interface%20Specification`
19 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FPOSIX`

GNAT 3.15p

This is the last public release of GNAT from AdaCore that uses the GNAT Modified General Public License[20].

GNAT 3.15p has passed the Ada Conformity Assessment Test Suite[21] (ACATS[22]). It was released in October 2002.

The binary distribution from AdaCore also contains an Ada-aware version of the GNU Debugger (GDB[23]), and a graphical front-end to GDB called the GNU Visual Debugger (GVD).

Languages supported	Ada 83, Ada 95, C
License for the run-time library	GNAT-modified GPL
Native platforms	GNU/Linux on i386 (with glibc 2.1 or later), Microsoft Windows on i386, OS/2 2.0 or later on i386, Solaris 2.5 or later on SPARC
Cross platforms	none
Compiler back-end	GCC 2.8.1
Available from	ftp://ftp.cs.kuleuven.ac.be/pub/Ada-Belgium/mirrors/gnu-ada/3.15p/
Support	None
Add-ons included	ASIS, Florist, GLADE, GDB, Gnatwin (on Windows only), GtkAda 1.2, GVD

GNAT Pro

GNAT Pro is the professional version of GNAT, offered as a subscription package by AdaCore. The package also includes professional consulting, training and maintenance services. AdaCore can provide custom versions of the compiler for native or cross development. For more information, see `http://www.adacore.com/`.

Languages supported	Ada 83, Ada 95, Ada 2005, Ada 2012, C, and optionally C++
License for the run-time library	GNAT-modified GPL
Native platforms	many, see `http://www.adacore.com/home/products/gnatpro/supported_platforms/`
Cross platforms	many, see `http://www.adacore.com/home/products/gnatpro/supported_platforms/`; even more on request

20 `http://en.wikipedia.org/wiki/GNAT%20Modified%20General%20Public%20License`
21 `http://en.wikipedia.org/wiki/ISO%2018009`
22 `http://en.wiktionary.org/wiki/ACATS`
23 `http://en.wikipedia.org/wiki/GDB`

Compiler back-end	GCC 4.3
Available from	`http://www.adacore.com/` by subscription (commercial)
Support	Commercial; customer-only bug database
Add-ons included	ASIS, Florist, GDB, GLADE, GPS, GtkAda, XML/Ada, and many more in source and, on request, binary form.

GCC

GNAT has been part of the Free Software Foundation[24]'s GCC[25] since October 2001. The Free Software Foundation does not distribute binaries, only sources. Its licensing of the run-time library for Ada (and other languages) allows the development of proprietary software without necessarily imposing the terms of the GPL[26].

Most GNU/Linux distributions and several distributions for other platforms include prebuilt binaries; see below.

For technical reasons, we recommend against using the Ada compilers included in GCC 3.1, 3.2, 3.3 and 4.0. Instead, we recommend using GCC 3.4, 4.1 or later, or one of the releases from AdaCore[27] (3.15p, GPL Edition or Pro).

Since October 2003, AdaCore merge most of their changes from GNAT Pro into GCC during Stage 1[28]; this happens once for each major release. Since GCC 3.4, AdaCore has gradually added support for revised language standards, first Ada 2005 and now Ada 2012.

GCC version 4.4 switched to version 3 of the GNU General Public License[29] and grants a Runtime Library Exception[30] similar in spirit to the GNAT Modified General Public License[31] used in all previous versions. This Runtime Library Exception applies to run-time libraries for all languages, not just Ada.

As of GCC 4.7, released on 2012-03-22:

Languages supported	Ada 83, Ada 95, Ada 2005, parts of Ada 2012, C, C++, Fortran 95, Java, Objective-C, Objective-C++ (and others)
License for the run-time library	GPL version 3[32] with Runtime Library Exception[33]
Native platforms	none (source only)

24 `http://www.fsf.org/`
25 `http://gcc.gnu.org/`
26 `http://en.wikipedia.org/wiki/GNU%20General%20Public%20License`
27 `http://www.adacore.com`
28 `http://gcc.gnu.org/develop.html#stage1`
29 `http://www.gnu.org/licenses/gpl.html`
30 `http://www.gnu.org/licenses/gcc-exception.html`
31 `http://en.wikipedia.org/wiki/GNAT%20Modified%20General%20Public%20License`
32 `http://www.gnu.org/licenses/gpl.html`
33 `http://www.gnu.org/licenses/gcc-exception.html`

Cross platforms	none (source only)
Compiler back-end	GCC 4.7
Available from	`http://gcc.gnu.org/` in source only form.
Support	Volunteer; public bug database
Add-ons included	none

2.4.3 The GNU Ada Project

The GNU Ada Project[34] provides source and binary packages of various GNAT versions for several operating systems, and, importantly, the scripts used to create the packages. This may be helpful if you plan to port the compiler to another platform or create a cross-compiler; there are instructions for building your own GNAT compiler for GNU/Linux[35] and Mac OS X[36] users.

Both GPL[37] and GMGPL[38] or GCC Runtime Library Exception[39] versions of GNAT are available.

Languages supported	Ada 83, Ada 95, Ada 2005, C. (Some distributions also support Ada 2012, Fortran 90, Java, Objective C and Objective C++)
License for the run-time library	pure, GNAT-modified GPL, or GCC Runtime Library Exception
Native platforms	Fedora Core 4 and 5, MS-DOS, OS/2, Solaris 10, SuSE 10, MacOS X, (more?)
Cross platforms	none
Compiler back-end	GCC 2.8.1, 3.4, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 (various binary packages)
Available from	Sourceforge[40]
Support	Volunteer; public bug database
Add-ons included	AdaBrowse, ASIS, Booch Components, Charles, GPS, GtkAda (more?)

2.4.4 A# (A-Sharp, a.k.a. Ada for .NET)

This compiler is historical as it has now been merged into GNAT GPL Edition[41] and GNAT Pro[42].

34 `http://gnuada.sourceforge.net`
35 `http://ada.krischik.com/index.php/Articles/CompileGNAT`
36 `http://forward-in-code.blogspot.com/2011/11/building-gcc-again.html`
37 `http://en.wikipedia.org/wiki/GNU%20General%20Public%20License`
38 `http://en.wikipedia.org/wiki/GNAT%20Modified%20General%20Public%20License`
39 `http://www.gnu.org/licenses/gcc-exception.html`
40 `http://sourceforge.net/projects/gnuada/files/`
41 Chapter 2.4.1 on page 11
42 Chapter 2.4.2 on page 12

A# is a port of Ada to the .NET Platform[43]. A# was originally developed at the Department of Computer Science at the United States Air Force Academy which distribute A# as a service to the Ada community under the terms of the GNU general public license. A# integrates well with Microsoft Visual Studio 2005, AdaGIDE and the RAPID open-source GUI Design tool. As of 2006-06-06:

Languages supported	Ada 83, Ada 95, C
License for the run-time library	pure GPL
Native platforms	Microsoft .NET
Cross platforms	none
Compiler back-end	GCC 3.4 (GNAT GPL 2006 Edition?)
Available from	`http://sourceforge.net/projects/asharp/`
Support	None (but see GNAT Pro)
Add-ons included	none.

2.4.5 GNAT for AVR microcontrollers

Rolf Ebert and others provide a version of GNAT configured as a cross-compiler to various AVR microcontrollers[44], as well as an experimental Ada run-time library suitable for use on the microcontrollers. As of Version 1.1.0 (2010-02-25):

Languages supported	Ada 83, Ada 95, Ada 2005, C
License for the run-time library	GNAT-Modified GPL
Host platforms	GNU/Linux and Microsoft Windows on i386
Target platforms	Various AVR 8-bit microcontrollers
Compiler back-end	GCC 4.3
Available from	`http://avr-ada.sourceforge.net/`
Support	Volunteer; public bug database
Add-ons included	partial Ada run time system, AVR peripherals support library

2.4.6 GNAT for LEON

The Real-Time Research Group of the Technical University of Madrid (UPM, *Universidad Politécnica de Madrid*) wrote a Ravenscar[45]-compliant real-time kernel for execution on LEON processors[46] and a modified run-time library. They also provide a GNAT cross-compiler. As of version 2.0.1:

43 `http://www.microsoft.com/net/`
44 `http://en.wikipedia.org/wiki/Atmel_AVR`
45 `http://en.wikipedia.org/wiki/Ravenscar%20profile`
46 `http://en.wikipedia.org/wiki/LEON`

Languages supported	Ada 83, Ada 95, Ada 2005, C
License for the run-time library	pure GPL
Native platforms	none
Cross platforms	GNU/Linux on i686 to LEON2 bare boards
Compiler back-end	GCC 4.1 (GNAT GPL 2007 Edition)
Available from	`http://www.dit.upm.es/ork/`
Support	?
Add-ons included	OpenRavenscar real-time kernel; minimal run-time library

2.4.7 GNAT for Macintosh (Mac OS X)

GNAT for Macintosh[47] provides both FSF (GMGPL) and AdaCore (GPL) versions of GNAT[48] with Xcode[49] and Carbon[50] integration and bindings.

Note that this site was last updated for GCC 4.3 and Mac OS X Leopard (both PowerPC and Intel-based). Aside from the work on integration with Apple's Carbon graphical user interface and with Xcode 3.1 it may be preferable to see above[51].

There is also support at MacPorts[52]; the last update (at 25 Nov 2011) was for GCC 4.4.2.

2.4.8 Prebuilt packages as part of larger distributions

Many distributions contain prebuilt binaries of GCC or various public releases of GNAT from AdaCore. Quality varies widely between distributions. The list of distributions below is in alphabetical oder. *(Please keep it that way.)*

AIDE (for Microsoft Windows)

AIDE — Ada Instant Development Environment[53] is a complete one-click, just-works Ada distribution for Windows, consisting of GNAT, comprehensive documentation, tools and libraries. All are precompiled, and source code is also available. The installation procedure is particularly easy. AIDE is intended for beginners and teachers, but can also be used by advanced users.

Languages supported	Ada 83, Ada 95, C
License for the run-time library	GNAT-modified GPL

47 `http://www.macada.org/`
48 `http://en.wikipedia.org/wiki/GNAT`
49 `http://en.wikipedia.org/wiki/Xcode`
50 `http://en.wikipedia.org/wiki/Carbon%20%28API%29`
51 Chapter 2.4.3 on page 14
52 `https://trac.macports.org/browser/trunk/dports/lang/gnat-gcc`
53 `http://sr.sriviere.info/aide/aide.html`

Native platforms	Microsoft Windows on i386
Cross platforms	none
Compiler back-end	GCC 2.8.1
Available from	`http://www.ada-france.org/AIDE/`ftp://ftp.cs.kuleuven.ac.be/pub/Ada-Belgium/mirrors/aide/
Support	?
Add-ons included	ASIS, GDB, GPS, GtkAda (more?)

Blastwave (for Solaris on SPARC and x86)

Blastwave[54] has binary packages of GCC 3.4.5 and 4.0.2 with Ada support. The package names are gcc3ada and gcc4ada respectively.

Languages supported	Ada 83, Ada 95, parts of Ada 2005, C, C++, Fortran 95, Java, Objective-C, Objective-C++
License for the run-time library	GNAT-modified GPL
Native platforms	Solaris and OpenSolaris on SPARC
Cross platforms	none
Compiler back-end	GCC 3.4.5 and 4.0.2 (both available)
Support	?
Available from	`http://www.blastwave.org/`
Add-ons included	none (?)

OpenCSW (for Solaris on SPARC and x86)

OpenCSW[55] has binary packages of GCC 3.4.6 and 4.6.2 with Ada support. The package names are gcc3ada and gcc4ada respectively.

Languages supported	Ada 83, Ada 95, parts of Ada 2005, C, C++, Fortran 95, Java, Objective-C, Objective-C++
License for the run-time library	GNAT-modified GPL
Native platforms	Oracle Solaris and OpenSolaris on SPARC and x86
Cross platforms	none
Compiler back-end	GCC 3.4.6 and 4.6.2 (both available)
Support	?
Available from	`http://www.opencsw.org/`
Add-ons included	none (?)

54 `http://www.blastwave.org`
55 `http://www.opencsw.org`

Cygwin (for Microsoft Windows)

Cygwin[56], the Linux-like environment for Windows, also contains a version of the GNAT[57] compiler. The Cygwin[58] version of GNAT[59] is older than the MinGW[60] version and does not support DLLs and Multi-Threading (as of 11.2004).

Debian (GNU/Linux and GNU/kFreeBSD)

There is a Debian Policy for Ada[61] which tries to make Debian the best Ada development *and deployment* platform. The development platform includes the compiler and many libraries, pre-packaged and integrated so as to be easy to use in any program. The deployment platform is the renowned *stable*[62] distribution, which is suitable for mission-critical workloads and enjoys long life cycles, typically 3 to 4 years. Because Debian is a binary distribution, it is possible to deploy non-free, binary-only programs on it while enjoying all the benefits of a stable platform. Compiler choices are conservative for this reason, and the Policy mandates that all Ada programs and libraries be compiled with the same version of GNAT. This makes it possible to use all libraries in the same program. Debian separates run-time libraries from development packages, so that end users do not have to install the development system just to run a program.

The GNU Ada compiler can be installed on a Debian system with this command:

```
aptitude install gnat
```

This will also give you a list of related packages, which are likely to be useful for an Ada programmer.

Debian is unique in that it also allows programmers to use some of GNAT's internal components by means of two libraries:

- libgnatvsn (licensed under GNAT-Modified GPL) and
- libgnatprj (the project manager, licensed under pure GPL).

Debian packages make use of these libraries.

In the table below, the information about the future Debian 7.0 *Wheezy* is accurate as of July 2012 but may change.

56 http://www.cygwin.com
57 Chapter 2.4 on page 10
58 http://en.wikipedia.org/wiki/Cygwin
59 http://en.wikipedia.org/wiki/GNAT
60 http://en.wikipedia.org/wiki/MinGW
61 http://people.debian.org/~lbrenta/debian-ada-policy.html
62 http://www.debian.org/releases/stable/

	3.1 *Sarge*	**4.0 *Etch***	**5.0 *Lenny***	**6.0 *Squeeze***	**7.0 *Wheezy***
Release date	June 2005	April 2007	February 2009	February 2011	2013?
Languages supported	Ada 83, Ada 95, C	Ada 83, Ada 95, Ada 2005, parts of Ada 2012, C, C++, Fortran 95, Java, Objective-C, Objective-C++			
License for the run-time library	GNAT-modified GPL (both ZCX and SJLJ versions starting from 5.0 *Lenny*)			GPL version 3 with Run-time library exception	
Native platforms:	**3.1 *Sarge***	**4.0 *Etch***	**5.0 *Lenny***	**6.0 *Squeeze***	**7.0 *Wheezy***
`alpha`		yes	yes		
`amd64`		yes	yes	yes	yes
`armel`				preliminary	yes
`armhf`					yes
`hppa`		yes	yes	yes	
`hurd-i386`					preliminary
`i386`	yes	yes	yes	yes	yes
`ia64`		yes	yes	yes	yes
`kfreebsd-amd64`				yes	yes
`kfreebsd-i386`		yes	yes	yes	yes
`mips`		yes	yes	yes	yes
`mipsel`		yes	yes	yes	yes
`powerpc`	yes	yes	yes	yes	yes
`ppc64`			yes	yes	yes
`s390`		yes	yes	yes	yes
`sparc`	yes	yes	yes	yes	yes
Cross platforms	none				
Compiler back-end	GCC 2.8.1	GCC 4.1	GCC 4.3	GCC 4.4	GCC 4.6
Available from	`http://www.debian.org/`				

	3.1 *Sarge*	**4.0 *Etch***	**5.0 *Lenny***	**6.0 *Squeeze***	**7.0 *Wheezy***
Release date	June 2005	April 2007	February 2009	February 2011	2013?
Languages supported	Ada 83, Ada 95, C	Ada 83, Ada 95, Ada 2005, parts of Ada 2012, C, C++, Fortran 95, Java, Objective-C, Objective-C++			
License for the run-time library	GNAT-modified GPL (both ZCX and SJLJ versions starting from 5.0 *Lenny*)			GPL version 3 with Run-time library exception	
Native platforms:	**3.1 *Sarge***	**4.0 *Etch***	**5.0 *Lenny***	**6.0 *Squeeze***	**7.0 *Wheezy***
Support	Volunteer; public bug database; paid support available from third parties; public mailing list[63]				
Add-ons included	**3.1 *Sarge***	**4.0 *Etch***	**5.0 *Lenny***	**6.0 *Squeeze***	**7.0 *Wheezy***
ada-reference-manual	1995	1995	1995	2005	2012
AdaBindX	0.7.2				
AdaBrowse	4.0.2	4.0.2	4.0.2	4.0.3	4.0.3
AdaCGI	1.6	1.6	1.6	1.6	1.6
AdaControl		1.6r8	1.9r4	1.12r3	1.12r3
APQ (with PostgreSQL)				3.0	3.2
AdaSockets	1.8.4.7	1.8.4.7	1.8.4.7	1.8.8	1.8.10
Ahven			1.2	1.7	2.1
Alog			0.1	0.3	0.4.1
anet					0.1
ASIS	3.15p	2005	2007	2008	2010
AUnit	1.01	1.03	1.03	1.03	1.03
AWS	2.0	2.2	2.5 prerelease	2.7	2.10.2
Charles	2005-02-17	(superseded by Ada.Containers in gnat)			
Florist	3.15p	2006	2006	2009	2011

63 `http://lists.debian.org/debian-ada`

	3.1 *Sarge*	**4.0 *Etch***	**5.0 *Lenny***	**6.0 *Squeeze***	**7.0 *Wheezy***
Release date	June 2005	April 2007	February 2009	February 2011	2013?
Languages supported	Ada 83, Ada 95, C	Ada 83, Ada 95, Ada 2005, parts of Ada 2012, C, C++, Fortran 95, Java, Objective-C, Objective-C++			
License for the run-time library	GNAT-modified GPL (both ZCX and SJLJ versions starting from 5.0 *Lenny*)			GPL version 3 with Run-time library exception	
Native platforms:	**3.1 *Sarge***	**4.0 *Etch***	**5.0 *Lenny***	**6.0 *Squeeze***	**7.0 *Wheezy***
GDB	5.3	6.4	6.8	7.0.1	7.4.1
GLADE	3.15p	2006		(see PolyORB)	
GMPAda				0.0.20091124	0.0.20120331
GNADE	1.5.1	1.6.1	1.6.1	1.6.2	1.6.2
GNAT Checker	1999-05-19	(superseded by AdaControl)			
GPRBuild				1.3.0w	2011
GPS	2.1	4.0.1	4.0.1	4.3	5.0
GtkAda	2.4	2.8.1	2.8.1	2.14.2	2.24.0
Log4Ada				1.0	1.2
Narval				1.10.2	
OpenToken	3.0b	3.0b	3.0b	4.0b	4.0b
PC/SC Ada				0.6	0.7.1
PolyORB				2.6 prerelease	2.8 prerelease
PLPlot			5.9.0	5.9.5	5.9.5
Templates Parser		10.0+20060522	11.1	11.5	11.6
TextTools	2.0.3	2.0.3	2.0.5	2.0.6	
XML/Ada	1.0	2.2	3.0	3.2	4.1
XML-EZ-out				1.06	1.06.1

The ADT plugin for Eclipse (see section ObjectAda from Aonix[64]) can be used with GNAT as packaged for Debian Etch. Specify "/usr" as the toolchain path.

DJGPP (for MS-DOS)

DJGPP has GNAT[65] as part of their GCC[66] distribution.

DJGPP[67] is a port of a comprehensive collection of GNU utilities to MS-DOS with 32-bit extensions, and is actively supported (as of 1.2005). It includes the whole GCC[68] compiler collection, that now includes Ada. See the DJGPP[69] website for installation instructions.

DJGPP programs run also in a DOS command box in Windows, as well as in native MS-DOS systems.

FreeBSD

FreeBSD[70]'s ports collection[71] contains GNAT GPL 2006 Edition (package gnat-2006), GNAT 3.15p, GCC 4.1, 4.2 and 4.3 with support for Ada. The usual way to install a package on FreeBSD is to compile it from source; not all add-ons are compatible with all versions of GNAT provided.

You can also use the Debian packages described above in a jail, thanks to Debian GNU/k-FreeBSD.

As of 2008-11-10:

Languages supported	Ada 83, Ada 95, parts of Ada 2005, C
License for the run-time library	both pure and modified GPL available
Native platforms	FreeBSD on i386 (more?)
Cross platforms	none
Compiler back-end	GCC 2.8.1, 3.4, 4.1, 4.2, 4.3
Available from	`http://www.freebsd.org`
Support	Volunteer; public bug database
Add-ons included	AdaBindX, AdaCurses, AdaSDL, AdaSockets, AFlex+AYACC, ASIS, AUnit, Booch Components, CBind, Florist, GLADE, GtkAda, SGL, XML/Ada (more?)

64 Chapter 2.4.8 on page 24
65 `http://en.wikipedia.org/wiki/GNAT`
66 `http://en.wikipedia.org/wiki/GCC`
67 `http://www.delorie.com/djgpp/`
68 `http://en.wikipedia.org/wiki/GCC`
69 `http://www.delorie.com/djgpp/`
70 `http://www.freebsd.org`
71 `http://www.freebsd.org/ports`

Gentoo GNU/Linux

The GNU Ada compiler can be installed on a Gentoo system using emerge:

```
emerge dev-lang/gnat
```

In contrast to Debian, Gentoo is primarily a source distribution, so many packages are available only in source form, and require the user to recompile them (using emerge).

Also in contrast to Debian, Gentoo supports several versions of GNAT in parallel on the same system. Be careful, because not all add-ons and libraries are available with all versions of GNAT.

Languages supported	Ada 83, Ada 95, Ada 2005, C (more?)
License for the run-time library	pure or GNAT-modified GPL (both available)
Native platforms	Gentoo GNU/Linux on amd64, powerpc and i386
Cross platforms	none
Compiler back-end	GCC 3.4, 4.1 (various binary packages)
Available from	`http://www.gentoo.org/` (see other Gentoo dev-ada[72] packages)
Support	Volunteer; public bug database
Add-ons included	AdaBindX, AdaBroker, AdaDoc, AdaOpenGL, AdaSockets, ASIS, AUnit, Booch Components, CBind, Charles, Florist, GLADE, GPS, GtkAda, XML/Ada

Mandriva Linux

The GNU Ada compiler can be installed on a Mandriva system with this command:

```
urpmi gnat
```

MinGW (for Microsoft Windows)

MinGW — Minimalist GNU for Windows[73] contains a version of the GNAT compiler.

The current version of MinGW (5.1.6) contains gcc-4.5.0. This includes a fully functional GNAT compiler. If the automatic downloader does not work correctly you can download the compiler directly: pick gcc-4.5.0-1 from MinGW/BaseSystem/GCC/Version4/

72 `http://es.znurt.org/dev-ada`
73 `http://mingw.sourceforge.net`

old instructions
The following list should help you with the installation. (I may have forgotten something — but this is wiki, just add to the list)

1. Install *MinGW-3.1.0-1.exe*
   a) extract *binutils-2.15.91-20040904-1.tar.gz*
   b) extract *mingw-runtime-3.5.tar.gz*
   c) extract *gcc-core-3.4.2-20040916-1.tar.gz*
   d) extract *gcc-ada-3.4.2-20040916-1.tar.gz*
   e) extract *gcc-g++-3.4.2-20040916-1.tar.gz (Optional)*
   f) extract *gcc-g77-3.4.2-20040916-1.tar.gz (Optional)*
   g) extract *gcc-java-3.4.2-20040916-1.tar.gz (Optional)*
   h) extract *gcc-objc-3.4.2-20040916-1.tar.gz (Optional)*
   i) extract *w32api-3.1.tar.gz*
2. Install *mingw32-make-3.80.0-3.exe (Optional)*
3. Install *gdb-5.2.1-1.exe (Optional)*
4. Install *MSYS-1.0.10.exe (Optional)*
5. Install *msysDTK-1.0.1.exe (Optional)*
   a) extract *msys-automake-1.8.2.tar.bz2 (Optional)*
   b) extract *msys-autoconf-2.59.tar.bz2 (Optional)*
   c) extract *msys-libtool-1.5.tar.bz2 (Optional)*

I have made good experience in using *D:\MinGW* as target directory for all installations and extractions.

Also noteworthy is that the Windows version for GNAT from Libre is also based on MinGW.

In gcc-3.4.2-release_notes.txt from MinGW site reads: *please* check that the files in the /lib/gcc/mingw32/3.4.2/adainclude and adalib directories are flagged as read-only. This attribute is necessary to prevent them from being deleted when using gnatclean to clean a project.

So be sure to do this.

SuSE Linux

All versions of SuSE Linux have a GNAT compiler included. SuSE versions 9.2 and higher also contains ASIS, Florist and GLADE libraries. The following two packages are needed:

```
gnat
gnat-runtime
```

For SuSE version 12.1, the compiler is in the package

```
gcc46-ada
libada46
```

For 64 bit system you will need the 32 bit compatibility packages as well:

```
gnat-32bit
gnat-runtime-32bit
```

Ubuntu

Ubuntu (and derivatives like Kubuntu, Xubuntu...) is a Debian-based Linux distribution, thus the installation process described above[74] can be used. Graphical package managers like Synaptic or Adept can also be employed to select the Ada packages.

2.5 ICC from Irvine Compiler Corporation

Irvine Compiler Corporation[75] provides native and cross compilers for various platforms.`http://www.irvine.com/products.html` The compiler and run-time system support development of certified, safety-critical software.

Commercial, proprietary. No-cost evaluation is possible on request. Royalty-free redistribution of the run-time system is allowed.

2.6 Janus/Ada 83 and 95 from RR Software

RR Software[76] offers native compilers for MS-DOS, Microsoft Windows and various Unix and Unix-like systems, and a library for Windows GUI programming called CLAW. There are academic, personal and professional editions, as well as support options.

Commercial but relatively cheap; proprietary.

2.7 MAXAda from Concurrent

Concurrent[77] offers MAXAda[78], an Ada 95 compiler for Linux/Xeon and PowerPC platforms, and Ada bindings to POSIX and X/Motif.`http://www.ccur.com/pdf/cpb-sw-maxada.pdf`

Commercial, proprietary.

74 Chapter 2.4.8 on page 18
75 `http://www.irvine.com/`
76 `http://www.rrsoftware.com`
77 `http://www.ccur.com/`
78 `http://www.ccur.com/products_rt_maxada.aspx`

2.8 ObjectAda from Atego (formerly Aonix)

Atego[79] offers native and cross compilers for various platforms. They come with an IDE, a debugger, a plug-in for Eclipse and a POSIX binding[80] `http://www.aonix.com/pdf/oa-linux.pdf`.

On Microsoft Windows and GNU/Linux on i386, Aonix offers two pricing models, at the customer's option: either a perpetual license fee with optional support, or just the yearly support fee: For Linux, that's $3000 for a single user or $12,000 for a 5-user service pack. See the full press release[81].

In addition, they offer "ObjectAda Special Edition": a no-cost evaluation version of ObjectAda that limits the size of programs that can be compiled with it, but is otherwise fully functional, with IDE and debugger. Free registration required[82].

A recent contribution by Atego is ADT[83] for Eclipse[84]. The *Ada Development Tools* add Ada language support to the Eclipse open source development platform. ADT can be used with Aonix compilers, and with GNAT. An open source vendor supported project is outlined for ADT at Eclipse[85]. Codenamed *Hibachi* and showcased at the Ada Conference UK 2007 and during Ada-Europe 2007, the project has now been officially created[86].

Commercial, proprietary.

2.9 PowerAda from OC Systems

OC Systems[87] offers Ada compilers and bindings to POSIX and X-11:

- PowerAda[88], an Ada 95 compiler for Linux and AIX,
- LegacyAda/390[89], an Ada 83 compiler for IBM System 370 and 390 mainframes

Commercial, proprietary.

2.10 Rational Apex from Atego (formerly IBM Rational[90])

Rational Apex[91] for native and embedded development.

79 `http://www.atego.com/`
80 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FPOSIX`
81 `http://www.atego.com/pressreleases/pressitem/aonix-shatters-ada-price-barrier-for-linux`
82 `http://www.atego.com/support/eval-aonix-objectada/`
83 `http://www.aonix.com/adt.html`
84 `http://www.eclipse.org`
85 `http://www.eclipse.org/proposals/adt/`
86 `http://www.eclipse.org/hibachi/`
87 `http://www.ocsystems.com/`
88 `http://www.ocsystems.com/prod_powerada.html`
89 `http://www.ocsystems.com/prod_legacyada.html`
90 Atego acquires IBM Rational Apex Ada Developer product family
91 `http://www-306.ibm.com/software/awdtools/developer/ada/`

Commercial, proprietary.

2.11 SCORE from DDC-I

SCORE from DDC-I

DDC-I[92] offers its SCORE cross-compilers for embedded development. SCORE stands for Safety-Critical, Object-oriented, Real-time Embedded.

Commercial, proprietary.

2.12 XD Ada from SWEP-EDS

XD Ada from SWEP-EDS

XD Ada[93] is an Ada 83 cross-compiler for embedded development. Hosts include VAX, Alpha and Integrity Servers running OpenVMS. Targets include Motorola 68000 and MIL-STD-1750A processors.

Commercial, proprietary.

2.13 XGC Ada from XGC Software

XGC Ada from XGC Software

XGC compilers are GCC with custom run-time libraries suitable for avionics and space applications. The run-time kernels are very small and do not support exception propagation (i.e. you can handle an exception only in the subprogram that raised it).

Commercial but some versions are also offered as free downloads. Free Software.

Languages supported	Ada 83, Ada 95, C
License for the run-time library	GNAT-Modified GPL
Native platforms	none

92 http://www.ddci.com/

93 http://www.swep-eds.com/XD%20Ada/Xd%20ada.htm

Cross platforms	Hosts: sun-sparc-solaris, pc-linux2.*; targets are bare boards with ERC32[94], MIL-STD-1750A[95], Motorola 68000[96] family or Intel 32-bit[97] processors. PowerPC[98] and Intel 80186[99] targets on request.
Compiler back-end	GCC 2.8.1
Available from	`http://www.xgc.com/`
Support	Commercial
Add-ons included	Ravenscar-compliant run-time kernels, certified for avionics and space applications; gdb cross-debugger; target simulator.

2.14 References

References

94 `http://en.wikipedia.org/wiki/ERC32`
95 `http://en.wikipedia.org/wiki/MIL-STD-1750A`
96 `http://en.wikipedia.org/wiki/Motorola_68000`
97 `http://en.wikipedia.org/wiki/IA_32`
98 `http://en.wikipedia.org/wiki/PowerPC`
99 `http://en.wikipedia.org/wiki/Intel_80186`

3 Building

Ada programs are usually easier to build than programs written in other languages like C or C++, which frequently require a makefile. This is because an Ada source file already specifies the dependencies of its source unit. See the **with** keyword[1] for further details.

Building an Ada program is not defined by the Reference Manual, so this process is absolutely dependent on the compiler. Usually the compiler kit includes a make tool which compiles a main program and all its dependencies, and links an executable file.

3.1 Building with various compilers

Building with various compilers

This list is incomplete. You can help Wikibooks by adding the build information[2] for other compilers.

3.1.1 GNAT

With GNAT[3], you can run this command:

```
gnat make <your_unit_file>
```

If the file contains a procedure, gnatmake will generate an executable file with the procedure as main program. Otherwise, e.g. a package, gnatmake will compile the unit and all its dependencies.

GNAT command line

gnatmake can be written as one word `gnatmake` or two words `gnat make`. For a full list of gnat commands just type `gnat` without any command options. The output will look something like this:

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fwith
2 http://en.wikibooks.org/w/index.php?title=Ada_Programming/Building&action=edit
3 http://en.wikipedia.org/wiki/GNAT

```
GNAT 3.4.3 Copyright 1996-2004 Free Software Foundation, Inc.

List of available commands

GNAT BIND               gnatbind
GNAT CHOP               gnatchop
GNAT CLEAN              gnatclean
GNAT COMPILE            gnatmake -c -f -u
GNAT ELIM               gnatelim
GNAT FIND               gnatfind
GNAT KRUNCH             gnatkr
GNAT LINK               gnatlink
GNAT LIST               gnatls
GNAT MAKE               gnatmake
GNAT NAME               gnatname
GNAT PREPROCESS         gnatprep
GNAT PRETTY             gnatpp
GNAT STUB               gnatstub
GNAT XREF               gnatxref

Commands FIND, LIST, PRETTY, STUB and XREF accept project file
switches -vPx, -Pprj and -Xnam=val
```

For further help on the option just type the command (one word or two words — as you like) without any command options.

GNAT IDE

The GNAT toolchain comes with an IDE[4] called GPS[5]. You need to download and install it separately. The GPS features a graphical user interface[6].

There are also GNAT plugins for Emacs[7] (Ada Mode[8]), KDevelop[9] and Vim[10] (Ada Mode[11]) available.

Both Emacs and Vim Ada-Mode are maintained by The GNU Ada project[12].

GNAT with Xcode

Apple's free (gratis) IDE, Xcode, is included with every Macintosh but requires an explicit installation step from DVD-ROM or CD-ROM. It is also downloadable from `http://developer.apple.com/`. Xcode uses the GNU Compiler Collection[13] and thus supports Ada, GDB[14], etc... and also includes myriad tools for optimizing code which are unique to

4 http://en.wikipedia.org/wiki/Integrated%20development%20environment
5 http://en.wikipedia.org/wiki/GNAT%20Programming%20Studio
6 http://en.wikipedia.org/wiki/Graphical%20user%20interface
7 http://en.wikipedia.org/wiki/Emacs
8 http://stephe-leake.org/emacs/ada-mode/emacs-ada-mode.html
9 http://en.wikipedia.org/wiki/KDevelop
10 http://en.wikipedia.org/wiki/Vim%20%28text%20editor%29
11 http://www.vim.org/scripts/script.php?script_id=1609
12 http://gnuada.sourceforge.net
13 http://en.wikipedia.org/wiki/GNU%20Compiler%20Collection
14 http://en.wikipedia.org/wiki/GNU%20Debugger

the Macintosh platform. However, GNAT must be installed separately as it is (as of 2008) not distributed as part of Xcode. Get the binary and/or sources at `http://www.macada.org/`, along with numerous tools and bindings including bindings to Apple's Carbon frameworks which allow the development of complete, "real" Mac programs, all in Ada.

3.1.2 Rational APEX

Rational APEX is a complete development environment comprising a language sensitive editor, compiler, debugger, coverage analyser, configuration management and much more. You normally work with APEX running a GUI.

APEX has been built for the development of big programs. Therefore the basic entity of APEX is a *subsystem*, a directory with certain traits recognized by APEX. All Ada compilation units have to reside in subsystems.

You can define an *export set*, i.e. the set of Ada units visible to other subsystems. However for a subsystem A to gain visibility to another subsystem B, A has to *import* B. After importing, A sees all units in B's export set. (This is much like the with-clauses, but here visibility means only potential visibility for Ada: units to be actually visible must be mentioned in a with-clause of course; units not in the export set cannot be used in with-clauses of Ada units in external subsystems.)

Normally subsystems should be hierarchically ordered, i.e. form a directed graph. But for special uses, subsystems can also mutually import one another.

For configuration management, a subsystem is decomposed in *views*, subdirectories of the subsystem. Views hold different development versions of the Ada units. So actually it's not subsystems which import other subsystems, rather subsystem views import views of other subsystems. (Of course, the closure of all imports must be consistent — it cannot be the case that e.g. subsystem (A, view A1) imports subsystems (B, B1) and (C, C1), whereas (B, B1) imports (C, C2)).

A view can be defined to be the development view. Other views then hold releases at different stages.

Each Ada compilation unit has to reside in a file of its own. When compiling an Ada unit, the compiler follows the with-clauses. If a unit is not found within the subsystem holding the compile, the compiler searches the import list (only the direct imports are considered, not the closure).

Units can be taken under version control. In each subsystem, a set of *histories* can be defined. An Ada unit can be taken under control in a history. If you want to edit it, you first have to check it out — it gets a new version number. After the changes, you can check it in again, i.e. make the changes permanent (or you abandon your changes, i.e. go back to the previous version). You normally check out units in the development view only; check-outs in release views can be forbidden.

You can select which version shall be the active one; normally it is the one latest checked in. You can even switch histories to get different development paths. e.g. different bodies of the same specification for different targets.

3.1.3 ObjectAda

ObjectAda is a set of tools for editing, compiling, navigating and debugging programs written in Ada. There are various editions of ObjectAda. With some editions you compile programs for the same platform and operating systems on which you run the tools. These are called native. With others, you can produce programs for different operating systems and platforms. One possible platform is the Java virtual machine.

These remarks apply to the native Microsoft Windows edition. You can run the translation tools either from the IDE or from the command line.

Whether you prefer to work from the IDE, or from the command line, a little bookkeeping is required. This is done by creating a project. Each project consists of a number of source files, and a number of settings like search paths for additional Ada libraries and other dependences. Each project also has at least one target. Typically, there is a debug target, and a release target. The names of the targets indicate their purpose. At one time you compile for debugging, typically during development, at other times you compile with different settings, for example when the program is ready for release. Some (all commercial?) editions of ObjectAda permit a Java (VM) target.

3.1.4 DEC Ada for VMS

DEC Ada is an Ada 83 compiler for VMS[15]. While “DEC Ada” is probably the name most users know, the compiler is now called “ HP Ada[16]”. It had previously been known also by names of "VAX Ada" and "Compaq Ada".

DEC Ada uses a true library management system — so the first thing you need to do is create and activate a library:

```
ACS Library Create [MyLibrary]
ACS Set Library [MyLibrary]
```

When creating a library you already set some constraints like support for Long_Float or the available memory size. So carefully read

```
HELP ACS Library Create *
```

Then next step is to load your Ada sources into the library:

```
ACS Load [Source]*.ada
```

The sources don't need to be perfect at this stage but syntactically correct enough for the compiler to determine the packages declared and analyze the **with** statements. Dec Ada allows you to have more than one package in one source file and you have any filename

15 http://en.wikipedia.org/wiki/OpenVMS
16 http://h71000.www7.hp.com/commercial/ada/ada_index.html

convention you like. The purpose of `ACS Load` is the creation of the dependency tree between the source files.

Next you compile them:

```
ACS Compile *
```

Note that compile take the package name and not the filename. The wildcard `*` means *all packages loaded*. The compiler automatically determines the right order for the compilation so a make[17] tool is not strictly needed.

Last but not least you link your file into an

```
ACS Link /Executable=[Executables]Main.exe Main
```

On large systems you might want to break sources down into several libraries — in which case you also need

```
ACS Merge /Keep *
```

to merge the content of the current library with the library higher up the hierarchy. The larger libraries should then be created with:

```
ACS Library Create /Large
```

This uses a different directory layout more suitable for large libraries.

DEC Ada IDE

Dec Ada comes without an IDE, however the DEC LSE[18] as well as the Ada Mode[19] of the Vim text editor[20] support DEC Ada.

3.2 Compiling our Demo Source

Compiling our Demo Source

Once you have downloaded[21] our example programs you might wonder how to compile them.

17 http://en.wikipedia.org/wiki/make%20%28software%29
18 http://en.wikipedia.org/wiki/Language-Sensitive_Editor
19 http://www.vim.org/scripts/script.php?script_id=1609
20 http://en.wikipedia.org/wiki/Vim%20%28text%20editor%29
21 https://sourceforge.net/project/showfiles.php?group_id=124904

First you need to extract the sources. Use your favorite zip tool[22] to achieve that. On extraction a directory with the same name as the filename is created. Beware: WinZip might also create a directory equaling the filename so Windows users need to be careful using the right option otherwise they end up with *wikibook-ada-1_2_0.src\wikibook-ada-1_2_0*.

Once you extracted the files you will find all sources in *wikibook-ada-1_2_0/Source*. You could compile them right there. For your convenience we also provide ready made project files for the following IDEs (If you find a directory for an IDEs not named it might be in the making and not actually work).

3.2.1 GNAT

You will find multi-target GNAT Project files and a multi-make Makefile file in *wikibook-ada-2_0_0/GNAT*. For i686 Linux and Windows you can compile any demo using:

```
gnat make -P project_file
```

You can also open them inside the GPS with

```
gps -P project_file
```

For other target platform it is a bit more difficult since you need to tell the project files which target you want to create. The following options can be used:

style ("Debug", "Release")

you can define if you like a debug or release version so you can compare how the options affect size and speed.

os ("Linux", "OS2", "Windows_NT", "VMS")

choose your operating system. Since there is no Ada 2005 available for OS/2 don't expect all examples to compile.

target ("i686", "x86_64", "AXP")

choose your CPU — "i686[23]" is any form of 32bit Intel or AMD CPU, "x86_64[24]" is an 64 bit Intel or AMD CPU and if you have an "AXP[25]" then you know it.

Remember to type all options as they are shown. To compile a debug version on x86-64 Linux you type:

```
gnat make -P project_file -Xstyle=Debug -Xos=Linux -Xtarget=x86_64
```

22 http://en.wikipedia.org/wiki/ZIP%20%28file%20format%29
23 http://en.wikipedia.org/wiki/x86
24 http://en.wikipedia.org/wiki/x86-64
25 http://en.wikipedia.org/wiki/DEC%20Alpha

As said in the beginning there is also a **makefile** available that will automatically determine the target used. So if you have a GNU make you can save yourself a lot of typing by using:

```
make project
```

or even use

```
make all
```

to make all examples in debug and release in one go.

Each compile is stored inside its own directory which is created in the form of *wikibook-ada-2_0_0/GNAT/**OS-Target-Style***. Empty directories are provided inside the archive.

3.2.2 Rational APEX

APEX uses the subsystem and view directory structure, so you will have to create those first and copy the source files into the view. After creating a view using the architecture model of your choice, use the menu option "Compile -> Maintenance -> Import Text Files". In the Import Text Files dialog, add "wikibook-ada-2_0_0/Source/*.ad?" to select the Ada source files from the directory you originally extracted to. Apex uses the file extensions .1.ada for specs and .2.ada for bodies — don't worry, the import text files command will change these automatically.

To link an example, select its main subprogram in the directory viewer and click the link button in the toolbar, or "Compile -> Link" from the menu. Double-click the executable to run it. You can use the shift-key modifier to bypass the link or run dialog.

3.2.3 ObjectAda

ObjectAda command-line

The following describes using the ObjectAda tools for Windows in a console window.

Before you can use the ObjectAda tools from the command line, make sure the `PATH` environment variable lists the directory containing the ObjectAda tools. Something like

```
set path=%path%;P:\Programs\Aonix\ObjectAda\bin
```

A minimal ObjectAda project can have just one source file. like the Hello World program provided in

File: hello_world_1.adb

To build an executable from this source file, follow these steps (assuming the current directory is a fresh one and contains the above mentioned source file):

- Register your source files:

```
X:\some\directory> adareg hello_world_1.adb
```

This makes your sources known to the ObjectAda tools. Have a look at the file UNIT.MAP created by adareg in the current directory if you like seeing what is happening under the hood.

- Compile the source file:

```
X:\some\directory> adacomp hello_world_1.adb
Front end of hello_world_1.adb succeeded with no errors.
```

- Build the executable program:

```
X:\some\directory> adabuild hello_world_1
ObjectAda Professional Edition Version 7.2.2: adabuild
   Copyright (c) 1997-2002 Aonix.  All rights reserved.
Linking...
Link of hello completed successfully
```

Notice that you specify the name of the main unit as argument to `adabuild`, not the name of the source file. In this case, it is *Hello_World_1* as in

```
procedure Hello_World_1 is
```

More information about the tools can be found in the user guide *Using the command line interface*, installed with the ObjectAda tools.

3.3 External links

External links

- GNAT Online Documentation:
  - GNAT User's Guide[26]
- DEC Ada:
  - Developing Ada Products on OpenVMS[27] (PDF)
  - DEC Ada — Language Reference Manual[28] (PDF)
  - DEC Ada — Run-Time Reference[29] (PDF)

26 http://gcc.gnu.org/onlinedocs/gcc-4.0.1/gnat_ugn_unw/
27 http://h71000.www7.hp.com/commercial/ada/ada_dap.pdf
28 http://h71000.www7.hp.com/commercial/ada/ada_lrm.pdf
29 http://h71000.www7.hp.com/commercial/ada/ada_rtr.pdf

4 Control Statements

4.1 Conditionals

Conditionals

Conditional clauses are blocks of code that will only execute if a particular expression (the condition) is true[1].

4.1.1 *if-else*

The *if-else* statement is the simplest of the conditional statements. They are also called branches, as when the program arrives at an *if* statement during its execution, control will "branch" off into one of two or more "directions". An *if-else* statement is generally in the following form:

```
if condition then
    statement;
else
    other statement;
end if;
```

If the original condition is met, then all the code within the first statement is executed. The optional else section specifies an alternative statement that will be executed if the condition is false. Exact syntax will vary between programming languages, but the majority of programming languages (especially procedural[2] and structured[3] languages) will have some form of if-else conditional statement built-in. The if-else statement can usually be extended to the following form:

```
if condition then
    statement;
elsif condition then
    other statement;
elsif condition then
    other statement;
...
else
    another statement;
end if;
```

1 http://en.wikipedia.org/wiki/Truth%20function
2 http://en.wikibooks.org/wiki/Computer%20Programming%2FProcedural%20programming
3 http://en.wikibooks.org/wiki/Computer%20Programming%2FStructured%20programming

Only one statement in the entire block will be executed. This statement will be the first one with a condition which evaluates to be true. The concept of an if-else-if structure is easier to understand with the aid of an example:

```
with Ada ;
use  Ada ;
...
type Degrees is new Float range -273.15 .. Float'Last;
...
Temperature : Degrees;
...
if Temperature >= 40.0 then
    Put_Line ("Wow!");
    Put_Line ("It's extremely hot");
elsif Temperature >= 30.0 then
    Put_Line ("It's hot");
elsif Temperature >= 20.0 then
    Put_Line ("It's warm");
elsif Temperature >= 10.0 then
    Put_Line ("It's cool");
elsif Temperature >= 0.0 then
    Put_Line ("It's cold");
else
    Put_Line ("It's freezing");
end if;
```

4.1.2 Optimizing hints

When this program executes, the computer will check all conditions in order until one of them matches its concept of truth. As soon as this occurs, the program will execute the statement immediately following the condition and continue on, without checking any other condition for truth. For this reason, when you are trying to optimize[4] a program, it is a good idea to sort your if-else conditions in descending probability[5]. This will ensure that in the most common scenarios, the computer has to do less work, as it will most likely only have to check one or two "branches" before it finds the statement which it should execute. However, when writing programs for the first time, try not to think about this too much lest you find yourself undertaking premature optimization[6].

Having said all that, you should be aware that an optimizing compiler[7] might rearrange your *if statement* at will when the statement in question is free from side effects[8]. Among other techniques optimizing compilers might even apply jump tables[9] and binary searches[10].

In Ada, conditional statements with more than one conditional do not use short-circuit evaluation by default. In order to mimic C/C++'s short-circuit evaluation, use **`and then`** or **`or else`** between the conditions.

4 http://en.wikipedia.org/wiki/Optimization_%2528computer_science%2529
5 http://en.wikibooks.org/wiki/Probability%2FIntroduction
6 http://en.wikipedia.org/wiki/Premature_optimization%23When_to_optimize
7 http://en.wikipedia.org/wiki/Optimizing_compiler
8 http://en.wikipedia.org/wiki/Side-effect_%28computer_science%29
9 http://en.wikipedia.org/wiki/Jump_table
10 http://en.wikipedia.org/wiki/Binary_search

4.1.3 *case*

Often it is necessary to compare one specific variable against several constant expressions. For this kind of conditional expression the *case* statement exists. For example:

```
case X is
   when 1 =>

      Walk_The_Dog;

   when 5 =>

      Launch_Nuke;

   when 8 | 10 =>

      Sell_All_Stock;

   when others =>

      Self_Destruct;

end case;
```

The subtype of X must be a discrete type, i.e. an enumeration or integer type.

In Ada, one advantage of the case statement is that the compiler will check the coverage of the choices, that is, all the values of the subtype of variable X must be present or a default branch **when others** must specify what to do in the remaining cases.

4.2 Unconditionals

Unconditionals

Unconditionals let you change the flow of your program without a condition. You should be careful when using unconditionals. Often they make programs difficult to understand. Read Isn't goto evil?[11] for more information.

4.2.1 *return*

End a function and return to the calling procedure or function.

For procedures:

```
return;
```

For functions:

11 Chapter 4.2 on page 39

```
return Value;
```

4.2.2 *goto*

Goto transfers control to the statement after the label.

```
   goto Label;

   Dont_Do_Something;

<< Label>>
   ...
```

Isn't *goto* evil?

One often hears that *goto* is **evil** and one should avoid using *goto*. But it is often overlooked that any return which is not the last statement inside a procedure or function is also an unconditional statement — a *goto* in disguise. There is an important difference though: a return is a forward only use of *goto*. Exceptions are also a type of goto statement; worse, they need not specify where they are going to!

Therefore if you have functions and procedures with more than one *return* statement you can just as well use *goto*. When it comes down to readability the following two samples are almost the same:

```
procedure Use_Return is
begin
   Do_Something;

   if Test then
      return;
   end if;

   Do_Something_Else;

   return;
end Use_Return;
```

```
procedure Use_Goto is
begin
   Do_Something;

   if Test then
      goto Exit_Use_Goto;
   end if;

   Do_Something_Else;

<< Exit_Use_Goto>>
   return;
end Use_Goto;
```

Because the use of a *goto* needs the declaration of a label, the *goto* is in fact twice as readable than the use of *return*. So if readability is your concern and not a strict "don't use *goto*"

programming rule then you should rather use *goto* than multiple *return*s. Best, of course, is the structured approach where neither *goto* nor multiple *returns* are needed:

```
procedure Use_If is
begin
   Do_Something;

   if not Test then

      Do_Something_Else;

   end if;

   return;
end Use_If;
```

4.3 Loops

Loops

Loops allow you to have a set of statements repeated over and over again.

4.3.1 Endless Loop

The endless loop is a loop which never ends and the statements inside are repeated forever. Never is meant as a relative term here — if the computer is switched off then even endless loops will end very abruptly.

```
Endless_Loop :
   loop

      Do_Something;

   end loop Endless_Loop;
```

The loop name (in this case, "Endless_Loop") is an optional feature of Ada. Naming loops is nice for readability but not strictly needed. Loop names are useful though if the program should jump out of an inner loop, see below.

4.3.2 Loop with condition at the beginning

This loop has a condition at the beginning. The statements are repeated as long as the condition is met. If the condition is not met at the very beginning then the statements inside the loop are never executed.

```
While_Loop :
   while X <= 5 loop

      X := Calculate_Something;

   end loop While_Loop;
```

4.3.3 Loop with condition at the end

This loop has a condition at the end and the statements are repeated until the condition is met. Since the check is at the end the statements are at least executed once.

```
Until_Loop :
   loop

      X := Calculate_Something;

      exit Until_Loop when X > 5;
   end loop Until_Loop;
```

4.3.4 Loop with condition in the middle

Sometimes you need to first make a calculation and exit the loop when a certain criterion is met. However when the criterion is not met there is something else to be done. Hence you need a loop where the exit condition is in the middle.

```
Exit_Loop :
   loop

      X := Calculate_Something;

      exit Exit_Loop when X > 5;

      Do_Something (X);

   end loop  Exit_Loop;
```

In Ada the **exit** condition can be combined with any other loop statement as well. You can also have more than one **exit** statement. You can also exit a named outer loop if you have several loops inside each other.

4.3.5 *for* loop

Quite often one needs a loop where a specific variable is counted from a given start value up or down to a specific end value. You could use the while[12] loop here — but since this is a very common loop there is an easier syntax available.

```
For_Loop :
   for I in Integer range 1 .. 10 loop

      Do_Something (I)

   end loop For_Loop;
```

You don't have to declare both type and range as seen in the example. If you leave out the type then the compiler will determine the type by context and leave out the range then the loop will iterate over every valid value for the type given.

12 Chapter 4.3.5 on page 42

As always with Ada: when "determine by context" gives two or more possible options then an error will be displayed and then you have to name the type to be used. Ada will only do "guess-works" when it is safe to do so.

for loop on arrays

Another very common situation is the need for a loop which iterates over every element of an array. The following sample code shows you how to achieve this:

```
Array_Loop :
   for I in X'Range loop

      X (I) := Get_Next_Element;

   end loop Array_Loop;
```

With X being an array. Note: This syntax is mostly used on arrays — hence the name — but will also work with other types when a full iteration is needed.

Unlike other loop counters, the loop counter i, in the for loop statement the value cannot be changed. The following is illegal.

```
for i in 1 .. 10 loop
   i := i + 1;
end loop;
```

Also the declaration of the loop counter ceases after the body of the loop.

Working Demo

The following Demo shows how to iterate over every element of an integer type.

File: range_1.adb

```
with Ada ;

procedure Range_1 is
   type Range_Type is range -5 .. 10;

   package T_IO renames Ada ;
   package I_IO is new  Ada  (Range_Type);

begin
   for A in Range_Type loop
      I_IO.Put (Item  => A,
                Width => 3,
                Base  => 10);

      if A < Range_Type'Last then
         T_IO.Put (",");
      else
         T_IO.New_Line;
      end if;
```

```
   end loop;
end Range_1;
```

4.4 See also

See also

4.4.1 Wikibook

- Ada Programming[13]

4.4.2 Ada Reference Manual

- 5.3 If Statements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-5-3.html}
- 5.4 Case Statements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-5-4.html}
- 5.5 Loop Statements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-5-5.html}
- 5.6 Block Statements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-5-6.html}
- 5.7 Exit Statements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-5-7.html}
- 5.8 Goto Statements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-5-8.html}
- 6.5 Return Statements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-6-5.html}

13 http://en.wikibooks.org/wiki/Ada%20Programming

5 Type System

Ada's type system allows the programmer to construct powerful abstractions that represent the real world, and to provide valuable information to the compiler, so that the compiler can find many logic or design errors before they become bugs. It is at the heart of the language, and good Ada programmers learn to use it to great advantage. Four principles govern the type system:

- **Strong typing**: types are incompatible with one another, so it is not possible to mix apples and oranges. There are, however, ways to convert between types.
- **Static typing**: type checked while compiling, this allows type errors to be found earlier.
- **Abstraction**: types represent the real world or the problem at hand; not how the computer represents the data internally. There are ways to specify exactly how a type must be represented at the bit level, but we will defer that discussion to another chapter.
- **Name equivalence**, as opposed to *structural equivalence* used in most other languages. Two types are compatible if and only if they have the same name; *not* if they just happen to have the same size or bit representation. You can thus declare two integer types with the same ranges that are totally incompatible, or two record types with exactly the same components, but which are incompatible.

Types are incompatible with one another. However, each type can have any number of *subtypes*, which are compatible with one another, and with their base type.

5.1 Predefined types

Predefined types

There are several predefined types, but most programmers prefer to define their own, application-specific types. Nevertheless, these predefined types are very useful as interfaces between libraries developed independently. The predefined library, obviously, uses these types too.

These types are predefined in the Standard package:

Integer

This type covers at least the range $-2^{15}+1$ .. $+2^{15}-1$ (RM 3.5.4 (21) ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-4.html`}). The Standard also defines Natural and Positive subtypes of this type.

Float

There is only a very weak implementation requirement on this type (RM 3.5.7 (14) ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-7.html`});

most of the time you would define your own floating-point types, and specify your precision and range requirements.

Duration

A fixed point type[1] used for timing. It represents a period of time in seconds (RM A.1 (43) ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-1.html}).

Character

A special form of Enumerations[2]. There are three predefined kinds of character types: 8-bit characters (called Character), 16-bit characters (called Wide_Character), and 32-bit characters (Wide_Wide_Character). Character has been present since the first version of the language (Ada 83[3]), Wide_Character was added in Ada 95[4], while the type Wide_-Wide_Character is available with Ada 2005[5].

String[6]

Three indefinite array types[7], of Character, Wide_Character, and Wide_Wide_Character respectively. The standard library contains packages for handling strings in three variants: fixed length (Ada), with varying length below a certain upper bound (Ada), and unbounded length (Ada). Each of these packages has a Wide_and a Wide_Wide_variant.

Boolean

A Boolean in Ada is an Enumeration[8] of False and True with special semantics.

Packages System and System predefine some types which are primarily useful for low-level programming and interfacing to hardware.

System.Address

An address in memory.

System.Storage_Elements.Storage_Offset

An offset, which can be added to an address to obtain a new address. You can also subtract one address from another to get the offset between them. Together, Address, Storage_Offset and their associated subprograms provide for address arithmetic.

System.Storage_Elements.Storage_Count

A subtype of Storage_Offset which cannot be negative, and represents the memory size of a data structure (similar to C's `size_t`).

System.Storage_Elements.Storage_Element

1 Chapter 10 on page 79
2 Chapter 8 on page 73
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%2083
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%2095
5 Chapter 23 on page 219
6 Chapter 15 on page 119
7 Chapter 11 on page 83
8 Chapter 8 on page 73

In most computers, this is a byte. Formally, it is the smallest unit of memory that has an address.

System.Storage_Elements.Storage_Array

An array of Storage_Elements without any meaning, useful when doing raw memory access.

5.2 The Type Hierarchy

The Type Hierarchy

Types are organized hierarchically. A type inherits properties from types above it in the hierarchy. For example, all scalar types (integer, enumeration, modular, fixed-point and floating-point types) have operators[9] "<", ">" and arithmetic operators defined for them, and all discrete types can serve as array indexes.

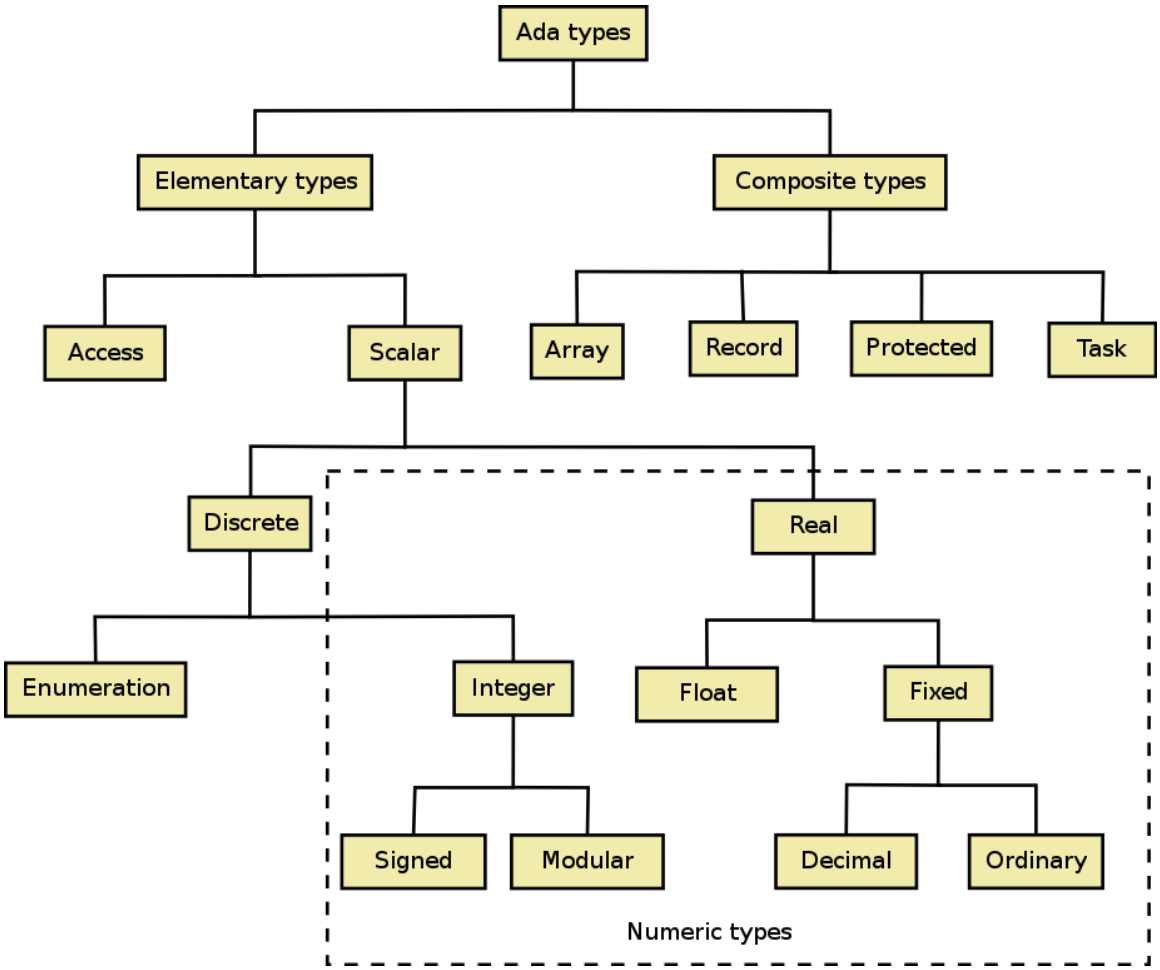

Figure 1 Ada type hierarchy

9 Chapter 37 on page 301

Here is a broad overview of each category of types; please follow the links for detailed explanations. Inside parenthesis there are equivalences in C and Pascal for readers familiar with those languages.

Signed Integers[10] (int, INTEGER)

Signed Integers are defined via the range[11] of values needed.

Unsigned Integers[12] (unsigned, CARDINAL)

Unsigned Integers are called Modular Types[13]. Apart from being unsigned they also have wrap-around functionality.

Enumerations[14] (enum, char, bool, BOOLEAN)

Ada Enumeration[15] types are a separate type family.

Floating point[16] (float, double, REAL)

Floating point types are defined by the digits[17] needed, the relative error bound.

Ordinary and Decimal Fixed Point[18] (DECIMAL)

Fixed point types are defined by their delta[19], the absolute error bound.

Arrays[20] ([ ], ARRAY [ ] OF, STRING)

Arrays with both compile-time and run-time determined size are supported.

Record[21] (struct, class, RECORD OF)

A **record** is a composite type[22] that groups one or more fields.

Access[23] (*, ^, POINTER TO)

Ada's Access[24] types may be more than just a simple memory address.

Task & Protected[25] (no equivalence in C or Pascal)

Task and Protected types allow the control of concurrency

Interfaces[26] (no equivalence in C or Pascal)

---

10 Chapter 6 on page 69
11 Chapter 6 on page 69
12 Chapter 7 on page 71
13 Chapter 7 on page 71
14 Chapter 8 on page 73
15 Chapter 8 on page 73
16 Chapter 9 on page 77
17 Chapter 9 on page 77
18 Chapter 10 on page 79
19 Chapter 10 on page 79
20 Chapter 11 on page 83
21 Chapter 12 on page 91
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes%23List%20of%20types
23 Chapter 13 on page 99
24 Chapter 13 on page 99
25 Chapter 21 on page 173
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Finterface

New in Ada 2005, these types are similar to the Java interfaces.

5.2.1 Classification of Types

The types of this hierarchy can be classified as follows.

Specific vs. Class-wide

```
type T is ...  -- specific
  T'Class      -- class-wide
```

Operations of specific types are *non-dispatching*, those on class-wide types are *dispatching*.

New types can be declared by *deriving* from specific types; primitive operations are *inherited* by derivation. You cannot derive from class-wide types.

Constrained vs. Unconstrained

```
type I is range 1 .. 10;           -- constrained
type AC is array (1 .. 10) of ...  -- constrained
```

```
type AU is array (I range <>) of ...          -- unconstrained
type R (X: Discriminant [:= Default]) is ...  -- unconstrained
```

By giving a *constraint* to an unconstrained subtype, a subtype or object becomes constrained:

```
subtype RC is R (Value);  -- constrained subtype of R
OC: R (Value);            -- constrained object of anonymous
constrained subtype of R
OU: R;                    -- unconstrained object
```

Declaring an unconstrained object is only possible if a default value is given in the type declaration above. The language does not specify how such objects are allocated. GNAT allocates the maximum size, so that size changes that might occur with discriminant changes present no problem. Another possibility is implicit dynamic allocation on the heap and deallocation followed be a re-allocation when the size changes.

Definite vs. Indefinite

```
type I is range 1 .. 10;                    -- definite
type RD (X: Discriminant := Default) is ...  -- definite
```

```
type T (<>) is ...                   -- indefinite
type AU is array (I range <>) of ...  -- indefinite
type RI (X: Discriminant) is ...      -- indefinite
```

Definite subtypes allow the declaration of objects without initial value, since objects of definite subtypes have constraints that are known at creation-time. Object declarations of indefinite subtypes need an initial value to supply a constraint; they are then constrained by

the constraint delivered by the initial value.

```
OT: T  := Expr;                       -- some initial expression
(object, function call, etc.)
OA: AU := (3 => 10, 5 => 2, 4 => 4);  -- index range is now 3 .. 5
OR: RI := Expr;                       -- again some initial
expression as above
```

Unconstrained vs. Indefinite

Note that unconstrained subtypes are not necessarily indefinite as can be seen above with RD: it is a definite unconstrained subtype.

5.3 Concurrency Types

Concurrency Types

The Ada language uses types for one more purpose in addition to classifying data + operations. The type system integrates concurrency (threading, parallelism). Programmers will use types for expressing the concurrent threads of control of their programs.

The core pieces of this part of the type system, the **task** types and the **protected** types are explained in greater depth in a section on tasking[27].

5.4 Limited Types

Limited Types

Limiting a type means disallowing assignment. The “concurrency types” described above are always limited. Programmers can define their own types to be limited, too, like this:

```
type T is limited ...;
```

(The ellipsis stands for **private**, or for a **record** definition, see the corresponding subsection on this page.) A limited type also doesn't have an equality operator unless the programmer defines one.

You can learn more in the limited types[28] chapter.

5.5 Defining new types and subtypes

27 Chapter 21 on page 173
28 Chapter 14 on page 113

You can define a new type with the following syntax:

```
type T is...
```

followed by the description of the type, as explained in detail in each category of type.

Formally, the above declaration creates a type and its first *subtype* named `T`. The type itself, correctly called the "type of T", is anonymous; the RM refers to it as *T* (in italics), but often speaks sloppily about the type T. But this is an academic consideration; for most purposes, it is sufficient to think of `T` as a type. For scalar types, there is also a base type called `T'Base`, which encompasses all values of T.

For signed integer types, the type of T comprises the (complete) set of mathematical integers. The base type is a certain hardware type, symmetric around zero (except for possibly one extra negative value), encompassing all values of T.

As explained above, all types are incompatible; thus:

```
type Integer_1 is range 1 .. 10;
type Integer_2 is range 1 .. 10;
A : Integer_1 := 8;
B : Integer_2 := A; --illegal!
```

is illegal, because `Integer_1` and `Integer_2` are different and incompatible types. It is this feature which allows the compiler to detect logic errors at compile time, such as adding a file descriptor to a number of bytes, or a length to a weight. The fact that the two types have the same range does not make them compatible: this is *name equivalence* in action, as opposed to structural equivalence. (Below, we will see how you can convert between incompatible types; there are strict rules for this.)

5.5.1 Creating subtypes

You can also create new subtypes of a given type, which will be compatible with each other, like this:

```
type Integer_1 is range 1 .. 10;
subtype Integer_2 is Integer_1      range 7 .. 11;  --bad
subtype Integer_3 is Integer_1'Base range 7 .. 11;  --OK
A : Integer_1 := 8;
B : Integer_3 := A; --OK
```

The declaration of `Integer_2` is bad because the constraint `7 .. 11` is not compatible with `Integer_1`; it raises `Contraint_Error` at subtype elaboration time.

`Integer_1` and `Integer_3` are compatible because they are both subtypes of the same type, namely `Integer_1'Base`.

It is not necessary that the subtype ranges overlap, or be included in one another. The compiler inserts a run-time range check when you assign A to B; if the value of A, at that point, happens to be outside the range of `Integer_3`, the program raises `Constraint_Error`.

There are a few predefined subtypes which are very useful:

```
subtype Natural  is Integer range 0 .. Integer'Last;
subtype Positive is Integer range 1 .. Integer'Last;
```

5.5.2 Derived types

A derived type is a new, full-blown type created from an existing one. Like any other type, it is incompatible with its parent; however, it inherits the primitive operations defined for the parent type.

```
type Integer_1 is range 1 .. 10;
type Integer_2 is new Integer_1 range 2 .. 8;
A : Integer_1 := 8;
B : Integer_2 := A; --illegal!
```

Here both types are discrete; it is mandatory that the range of the derived type be included in the range of its parent. Contrast this with subtypes. The reason is that the derived type inherits the primitive operations defined for its parent, and these operations assume the range of the parent type. Here is an illustration of this feature:

```
procedure Derived_Types is

   package Pak is
      type Integer_1 is range 1 .. 10;
      procedure P (I: in Integer_1); --primitive operation, assumes 1 ..  10
      type Integer_2 is new Integer_1 range 8 .. 10; --must not break P's assumption
      --procedure P (I: in Integer_2); inherited P implicitly defined here
   end Pak;

   package body Pak is
      --omitted
   end Pak;

   use Pak;
   A: Integer_1 := 4;
   B: Integer_2 := 9;

begin

   P (B); --OK, call the inherited operation

end Derived_Types;
```

When we call `P (B)`, the parameter B is converted to `Integer_1`; this conversion of course passes since the set of acceptable values for the derived type (here, 8 .. 10) must be included in that of the parent type (1 .. 10). Then P is called with the converted parameter.

Consider however a variant of the example above:

```
procedure Derived_Types is

  package Pak is
    type Integer_1 is range 1 .. 10;
    procedure P (I: in Integer_1; J: out Integer_1);
    type Integer_2 is new Integer_1 range 8 .. 10;
  end Pak;

  package body Pak is
    procedure P (I: in Integer_1; J: out Integer_1) is
    begin
      J := I - 1;
    end P;
  end Pak;

  use Pak;

  A: Integer_1 := 4;  X: Integer_1;
  B: Integer_2 := 8;  Y: Integer_2;

begin

  P (A, X);
  P (B, Y);

end Derived_Types;
```

When `P (B, Y)` is called, both parameters are converted to `Integer_1`. Thus the range check on J (7) in the body of P will pass. However on return parameter Y is converted back to `Integer_2` and the range check on Y will of course fail.

With the above in mind, you will see why in the following program Constraint_Error will be called at run time.

```
procedure Derived_Types is

  package Pak is
    type Integer_1 is range 1 .. 10;
    procedure P (I: in Integer_1; J: out Integer_1);
    type Integer_2 is new Integer_1'Base range 8 .. 12;
  end Pak;

  package body Pak is
    procedure P (I: in Integer_1; J: out Integer_1) is
    begin
      J := I - 1;
    end P;
  end Pak;

  use Pak;

  B: Integer_2 := 11;  Y: Integer_2;

begin

  P (B, Y);

end Derived_Types;
```

5.6 Subtype categories

Subtype categories

Ada supports various categories of subtypes which have different abilities. Here is an overview in alphabetical order.

5.6.1 Anonymous subtype

A subtype which does not have a name assigned to it. Such a subtype is created with a variable declaration:

```
X : String (1 .. 10) := (others => ' ');
```

Here, (1 .. 10) is the constraint. This variable declaration is equivalent to:

```
subtype Anonymous_String_Type is String (1 .. 10);

X : Anonymous_String_Type := (others => ' ');
```

5.6.2 Base type

In Ada, all types are anonymous[29] and only subtypes may be named[30]. For scalar types, there is a special subtype of the anonymous type, called the *base type*, which is nameable with the '*Base* attribute. The base type comprises all values of the *first subtype*. Some examples:

```
type Int is range 0 .. 100;
```

The base type `Int'Base` is a hardware type selected by the compiler that comprises the values of `Int`. Thus it may have the range -2^7 .. 2^7-1 or -2^{15} .. 2^{15}-1 or any other such type.

```
type Enum  is (A, B, C, D);
type Short is new Enum range A .. C;
```

`Enum'Base` is the same as `Enum`, but `Short'Base` also holds the literal `D`.

29 Chapter 5.11.1 on page 67
30 Chapter 5.6.6 on page 56

5.6.3 Constrained subtype

A subtype of an indefinite subtype[31] that adds constraints. The following example defines a 10 character string sub-type.

```
subtype String_10 is String (1 .. 10);
```

You cannot partially constrain an unconstrained subtype:

```
type My_Array is array (Integer range <>, Integer range <>) of Some_Type;

-- subtype Constr is My_Array (1 .. 10, Integer range <>);  illegal

subtype Constr is My_Array (1 .. 10, -100 .. 200);
```

Constraints for all indices must be given, the result is necessarily a definite subtype[32].

5.6.4 Definite subtype

A definite subtype[33] is a subtype whose size is known at compile-time. All subtypes which are not indefinite subtypes[34] are, by definition, definite subtypes[35].

Objects of definite subtypes may be declared without additional constraints.

5.6.5 Indefinite subtype

An **indefinite subtype** is a subtype whose size is not known at compile-time but is dynamically calculated at run-time. An indefinite subtype does not by itself provide enough information to create an object; an additional constraint or explicit initialization expression is necessary in order to calculate the actual size and therefore create the object.

```
X : String := "This is a string";
```

X is an object of the indefinite (sub)type String. Its constraint is derived implicitly from its initial value. X may change its value, but not its bounds.

It should be noted that it is not necessary to initialize the object from a literal. You can also use a function. For example:

31 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23indefinite_subtype
32 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23definite_subtype
33 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23definite_subtype
34 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23indefinite_subtype
35 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23definite_subtype

```
X : String := Ada.Command_Line.Argument (1);
```

This statement reads the first command-line argument and assigns it to X.

5.6.6 Named subtype

A subtype which has a name assigned to it. "First subtypes" are created with the keyword **type** (remember that types are always anonymous, the name in a type declaration is the name of the first subtype), others with the keyword **subtype**. For example:

```
type Count_To_Ten is range 1 .. 10;
```

Count_to_Ten is the first subtype of a suitable integer base type. However, if you would like to use this as an index constraint on String, the following declaration is illegal:

```
subtype Ten_Characters is String (Count_to_Ten);
```

This is because String has Positive as index, which is a subtype of Integer (these declarations are taken from package Standard):

```
subtype Positive is Integer range 1 .. Integer'Last;

type String is (Positive range <>) of Character;
```

So you have to use the following declarations:

```
subtype Count_To_Ten is Integer range 1 .. 10;
subtype Ten_Characters is String (Count_to_Ten);
```

Now Ten_Characters is the name of that subtype of String which is constrained to Count_-To_Ten. You see that posing constraints on types versus subtypes has very different effects.

5.6.7 Unconstrained subtype

A subtype of an indefinite subtype that does not add a constraint only introduces a new name for the original subtype.

```
subtype My_String is String;
```

`My_String` and `String` are interchangeable.

5.7 Qualified expressions

Qualified expressions

In most cases, the compiler is able to infer the type of an expression; for example:

```
type Enum is (A, B, C);
E : Enum := A;
```

Here the compiler knows that `A` is a value of the type Enum. But consider:

```
procedure Bad is
   type Enum_1 is (A, B, C);
   procedure P (E : in Enum_1) is... --omitted
   type Enum_2 is (A, X, Y, Z);
   procedure P (E : in Enum_2) is... --omitted
begin
   P (A); --illegal:  ambiguous
end Bad;
```

The compiler cannot choose between the two versions of P; both would be equally valid. To remove the ambiguity, you use a *qualified expression*:

```
   P (Enum_1'(A)); --OK
```

As seen in the following example, this syntax is often used when creating new objects. If you try to compile the example, it will fail with a compilation error since the compiler will determine that 256 is not in range of Byte.

File: convert_evaluate_as.adb

```
with Ada ;

procedure Convert_Evaluate_As is
   type Byte     is mod 2**8;
   type Byte_Ptr is access Byte;

   package T_IO renames Ada ;
   package M_IO is new Ada  (Byte);

   A : constant Byte_Ptr := new Byte'(256);
begin
   T_IO.Put ("A = ");
   M_IO.Put (Item  => A.all,
             Width =>  5,
             Base  => 10);
end Convert_Evaluate_As;
```

5.8 Type conversions

Type conversions

Data do not always come in the format you need them. You must, then, face the task of converting them. As a true multi-purpose language with a special emphasis on "mission critical", "system programming" and "safety", Ada has several conversion techniques. The most difficult part is choosing the right one, so the following list is sorted in order of utility. You should try the first one first; the last technique is a last resort, to be used if all others fail. There are also a few related techniques that you might choose instead of actually converting the data.

Since the most important aspect is not the result of a successful conversion, but how the system will react to an invalid conversion, all examples also demonstrate **faulty** conversions.

5.8.1 Explicit type conversion

An explicit type conversion looks much like a function call; it does not use the *tick* (apostrophe, ') like the qualified expression does.

```
Type_Name (Expression)
```

The compiler first checks that the conversion is legal, and if it is, it inserts a run-time check at the point of the conversion; hence the name *checked conversion*. If the conversion fails, the program raises Constraint_Error. Most compilers are very smart and optimise away the constraint checks; so, you need not worry about any performance penalty. Some compilers can also warn that a constraint check will always fail (and optimise the check with an unconditional raise).

Explicit type conversions are legal:

- between any two numeric types
- between any two subtypes of the same type
- between any two types derived from the same type (note special rules for tagged types)
- between array types under certain conditions (see RM 4.6(24.2/2..24.7/2))
- and *nowhere else*

(The rules become more complex with class-wide and anonymous access types.)

```
I: Integer := Integer (10);  --Unnecessary explicit type conversion
J: Integer := 10;            --Implicit conversion from universal integer
K: Integer := Integer'(10);  --Use the value 10 of type Integer:  qualified expression
                             --(qualification not necessary here).
```

This example illustrates explicit type conversions:

File: convert_checked.adb

```
with Ada ;

procedure Convert_Checked is
   type Short is range -128 .. +127;
   type Byte  is mod 256;

   package T_IO renames Ada ;
   package I_IO is new Ada  (Short);
   package M_IO is new Ada  (Byte);

   A : Short := -1;
   B : Byte;
begin
   B := Byte (A);  -- range check will lead to Constraint_Error
   T_IO.Put ("A = ");
   I_IO.Put (Item  =>  A,
             Width =>  5,
             Base  => 10);
   T_IO.Put (", B = ");
   M_IO.Put (Item  =>  B,
             Width =>  5,
             Base  => 10);
end Convert_Checked;
```

Explicit conversions are possible between any two numeric types: integers, fixed-point and floating-point types. If one of the types involved is a fixed-point or floating-point type, the compiler not only checks for the range constraints (thus the code above will raise Constraint_Error), but also performs any loss of precision necessary.

Example 1: the loss of precision causes the procedure to only ever print "0" or "1", since `P / 100` is an integer and is always zero or one.

```
with Ada.Text_IO;
procedure Naive_Explicit_Conversion is
   type Proportion is digits 4 range 0.0 .. 1.0;
   type Percentage is range 0 .. 100;
   function To_Proportion (P : in Percentage) return Proportion is
   begin
      return Proportion (P / 100);
   end To_Proportion;
begin
   Ada.Text_IO.Put_Line (Proportion'Image (To_Proportion (27)));
end Naive_Explicit_Conversion;
```

Example 2: we use an intermediate floating-point type to guarantee the precision.

```
with Ada.Text_IO;
procedure Explicit_Conversion is
   type Proportion is digits 4 range 0.0 .. 1.0;
   type Percentage is range 0 .. 100;
   function To_Proportion (P : in Percentage) return Proportion is
      type Prop is digits 4 range 0.0 .. 100.0;
   begin
      return Proportion (Prop (P) / 100.0);
   end To_Proportion;
begin
   Ada.Text_IO.Put_Line (Proportion'Image (To_Proportion (27)));
end Explicit_Conversion;
```

You might ask why you should convert between two subtypes of the same type. An example will illustrate this.

```
subtype String_10 is String (1 .. 10);
X: String := "A line long enough to make the example valid";
Slice: constant String := String_10 (X (11 .. 20));
```

Here, `Slice` has bounds 1 and 10, whereas `X (11 .. 20)` has bounds 11 and 20.

5.8.2 Change of Representation

Type conversions can be used for packing and unpacking of records or arrays.

```
type Unpacked is record
  --any components
end record;

type Packed is new Unpacked;
for  Packed use record
  --component clauses for some or for all components
end record;
```

```
P: Packed;
U: Unpacked;

P := Packed (U);  --packs U
U := Unpacked (P);  --unpacks P
```

5.8.3 Checked conversion for non-numeric types

The examples above all revolved around conversions between numeric types; it is possible to convert between any two numeric types in this way. But what happens between non-numeric types, e.g. between array types or record types? The answer is two-fold:

- you can convert explicitly between a type and types derived from it, or between types derived from the same type,
- and that's all. No other conversions are possible.

Why would you want to derive a record type from another record type? Because of representation clauses. Here we enter the realm of low-level systems programming, which is not for the faint of heart, nor is it useful for desktop applications. So hold on tight, and let's dive in.

Suppose you have a record type which uses the default, efficient representation. Now you want to write this record to a device, which uses a special record format. This special representation is more compact (uses fewer bits), but is grossly inefficient. You want to have a layered programming interface: the upper layer, intended for applications, uses the efficient representation. The lower layer is a device driver that accesses the hardware directly and uses the inefficient representation.

```
package Device_Driver is
   type Size_Type is range 0 .. 64;
   type Register is record
      A, B : Boolean;
      Size : Size_Type;
   end record;

   procedure Read (R : out Register);
   procedure Write (R : in Register);
end Device_Driver;
```

The compiler chooses a default, efficient representation for `Register`. For example, on a 32-bit machine, it would probably use three 32-bit words, one for A, one for B and one for Size. This efficient representation is good for applications, but at one point we want to convert the entire record to just 8 bits, because that's what our hardware requires.

```
package body Device_Driver is
   type Hardware_Register is new Register; --Derived type.
   for Hardware_Register use record
      A at 0 range 0 .. 0;
      B at 0 range 1 .. 1;
      Size at 0 range 2 .. 7;
   end record;

   function Get return Hardware_Register; --Body omitted
   procedure Put (H : in Hardware_Register); --Body omitted

   procedure Read (R : out Register) is
      H : Hardware_Register := Get;
   begin
      R := Register (H); --Explicit conversion.
   end Read;

   procedure Write (R : in Register) is
   begin
      Put (Hardware_Register (R)); --Explicit conversion.
   end Write;
end Device_Driver;
```

In the above example, the package body declares a derived type with the inefficient, but compact representation, and converts to and from it.

This illustrates that **type conversions can result in a change of representation**.

5.8.4 View conversion, in object-oriented programming

Within object-oriented programming[36] you have to distinguish between specific types and class-wide types.

With specific types, only conversions to ancestors are possible and, of course, are checked. During the conversion, you do not "drop" any components that are present in the derived type and not in the parent type; these components are still present, you just don't see them anymore. This is called a *view conversion*.

36 Chapter 22 on page 187

There are no conversions to derived types (where would you get the further components from?); *extension aggregates* have to be used instead.

```
type Parent_Type is tagged null record;
type Child_Type  is new Parent_Type with null record;

Child_Instance  : Child_Type;

--View conversion from the child type to the parent type:
Parent_View : Parent_Type := Parent_Type (Child_Instance);
```

Since, in object-oriented programming, an object of child type *is an* object of the parent type, no run-time check is necessary.

With class-wide types, conversions to ancestor and child types are possible and are checked as well. These conversions are also view conversions, no data is created or lost.

```
procedure P (Parent_View : Parent_Type'Class) is
   --View conversion to the child type:
   One : Child_Type := Child_Type (Parent_View);

   --View conversion to the class-wide child type:
   Two : Child_Type'Class := Child_Type'Class (Parent_View);
```

This view conversion involves a run-time check to see if `Parent_View` is indeed a view of an object of type `Child_Type`. In the second case, the run-time check accepts objects of type `Child_Type` but also any type derived from `Child_Type`.

View renaming

A renaming declaration does not create any new object and performs no conversion; it only gives a new name to something that already exists. Performance is optimal since the renaming is completely done at compile time. We mention it here because it is a common idiom in object oriented programming[37] to rename the result of a view conversion.

```
type Parent_Type is tagged record
   <components>;
end record;
type Child_Type is new Parent_Type with record
   <further components>;
end record;

Child_Instance : Child_Type;
Parent_View    : Parent_Type'Class renames Parent_Type'Class (Child_Instance);
```

Now, `Parent_View` is not a new object, but another name for `Child_Instance` viewed as the parent, i.e. only the parent components are visible, the further child components are hidden.

37 Chapter 22 on page 187

5.8.5 Address conversion

Ada's access type[38] is not just a memory location (a thin pointer). Depending on implementation and the access type[39] used, the access[40] might keep additional information (a fat pointer). For example GNAT keeps two memory addresses for each access[41] to an indefinite object — one for the data and one for the constraint informations (*Size*, *First*, *Last).*

If you want to convert an access to a simple memory location you can use the package System . Note however that an address and a fat pointer cannot be converted reversibly into one another.

The address of an array object is the address of its first component. Thus the bounds get lost in such a conversion.

```
type My_Array is array (Positive range <>) of Something;
A: My_Array (50 .. 100);

     A'Address = A(A'First)'Address
```

5.8.6 Unchecked conversion

One of the great criticisms of Pascal was "there is no escape". The reason was that sometimes you have to convert the incompatible. For this purpose, Ada has the generic function *Unchecked_Conversion*:

```
generic
   type Source (<>) is limited private;
   type Target (<>) is limited private;
function Ada  (S : Source) return Target;
```

Unchecked_Conversion will bit-copy the source data and reinterprete them under the target type without any checks. It is **your** chore to make sure that the requirements on unchecked conversion as stated in RM 13.9 `^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-13-9.html}` are fulfilled; if not, the result is implementation dependent and may even lead to abnormal data. Use the 'Valid attribute after the conversion to check the validity of the data in problematic cases.

A function call to (an instance of) *Unchecked_Conversion* will copy the source to the destination. The compiler may also do a conversion *in place* (every instance has the convention *Intrinsic*).

To use *Unchecked_Conversion* you need to instantiate the generic.

38 Chapter 13 on page 99
39 Chapter 13 on page 99
40 Chapter 13 on page 99
41 Chapter 13 on page 99

In the example below, you can see how this is done. When run, the example will output "A = -1, B = 255". No error will be reported, but is this the result you expect?

File: convert_unchecked.adb

```
with Ada ;
with Ada ;

procedure Convert_Unchecked is

   type Short is range -128 .. +127;
   type Byte  is mod 256;

   package T_IO renames Ada ;
   package I_IO is new Ada  (Short);
   package M_IO is new Ada  (Byte);

   function Convert is new Ada  (Source => Short,
                                                        Target =>
Byte);

   A : constant Short := -1;
   B : Byte;

begin

   B := Convert (A);
   T_IO.Put ("A = ");
   I_IO.Put (Item  =>  A,
             Width =>  5,
             Base  => 10);
   T_IO.Put (", B = ");
   M_IO.Put (Item  =>  B,
             Width =>  5,
             Base  => 10);

end Convert_Unchecked;
```

There is of course a range check in the assignment `B := Convert (A);`. Thus if B were defined as `B: Byte range 0 .. 10;`, Constraint_Error would be raised.

5.8.7 Overlays

If the copying of the result of *Unchecked_Conversion* is too much waste in terms of performance, then you can try overlays, i.e. address mappings. By using overlays, both objects share the same memory location. If you assign a value to one, the other changes as well. The syntax is:

```
for Target'Address use expression;
pragma  (Ada, Target);
```

where *expression* defines the address of the source object.

While overlays might look more elegant than Unchecked_Conversion, you should be aware that they are even more dangerous and have even greater potential for doing something very

wrong. For example if `Source'Size < Target'Size` and you assign a value to Target, you might inadvertently write into memory allocated to a different object.

You have to take care also of implicit initializations of objects of the target type, since they would overwrite the actual value of the source object. The Import pragma with convention Ada can be used to prevent this, since it avoids the implicit initialization, RM B.1 ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-1.html} .

The example below does the same as the example from "Unchecked Conversion".

File: convert_address_mapping.adb

```
with Ada ;

procedure Convert_Address_Mapping is
   type Short is range -128 .. +127;
   type Byte  is mod 256;

   package T_IO renames Ada ;
   package I_IO is new Ada  (Short);
   package M_IO is new Ada  (Byte);

   A : aliased Short;
   B : aliased Byte;

   for B'Address use A'Address;
  pragma  (Ada, B);

begin
   A := -1;
   T_IO.Put ("A = ");
   I_IO.Put (Item  =>  A,
             Width =>  5,
             Base  => 10);
   T_IO.Put (", B = ");
   M_IO.Put (Item  =>  B,
             Width =>  5,
             Base  => 10);
end Convert_Address_Mapping;
```

5.8.8 Export / Import

Just for the record: There is still another method using the **pragma** and **pragma** pragmas. However, since this method completely undermines Ada's visibility and type concepts even more than overlays, it has no place here in this language introduction and is left to experts.

5.9 Elaborated Discussion of Types for Signed Integer Types

Elaborated Discussion of Types for Signed Integer Types

As explained before, a type declaration

```
type T is range 1 .. 10;
```

declares an anonymous type *T* and its first subtype `T` (please note the italicization). *T* encompasses the complete set of mathematical integers. Static expressions and named numbers make use of this fact.

All numeric integer literals are of type `Universal_Integer`. They are converted to the appropriate specific type where needed. `Universal_Integer` itself has no operators.

Some examples with static named numbers:

```
S1: constant := Integer'Last + Integer'Last;       --"+" of Integer
S2: constant := Long_Integer'Last + 1;             --"+" of Long_Integer
S3: constant := S1 + S2;                           --"+" of root_integer
S4: constant := Integer'Last + Long_Integer'Last;  --illegal
```

Static expressions are evaluated at compile-time on the appropriate types with no overflow checks, i.e. mathematically exact (only limited by computer store). The result is then implicitly converted to `Universal_Integer`.

The literal 1 in `S2` is of type `Universal_Integer` and implicitly converted to `Long_Integer`.

`S3` implicitly converts the summands to `root_integer`, performs the calculation and converts back to `Universal_Integer`.

`S4` is illegal because it mixes two different types. You can however write this as

```
S5: constant := Integer'Pos (Integer'Last) + Long_Integer'Pos
(Long_Integer'Last);  --"+" of root_integer
```

where the Pos attributes convert the values to `Universal_Integer`, which are then further implicitly converted to `root_integer`, added and the result converted back to `Universal_Integer`.

`root_integer` is the anonymous greatest integer type representable by the hardware. It has the range `System.Min_Integer .. System.Max_Integer`. All integer types are rooted at `root_integer`, i.e. derived from it. `Universal_Integer` can be viewed as `root_integer'Class`.

During run-time, computations of course are performed with range checks and overflow checks on the appropriate subtype. Intermediate results may however exceed the range limits. Thus with `I, J, K` of the subtype `T` above, the following code will return the correct result:

```
I := 10;
J :=  8;
K := (I + J) - 12;
--I := I + J; -- range check would fail, leading to Constraint_Error
```

Real literals are of type `Universal_Real`, and similar rules as the ones above apply accordingly.

5.10 Relations between types

Relations between types

Types can be made from other types. Array types, for example, are made from two types, one for the arrays' index and one for the arrays' components. An array, then, expresses an association, namely that between one value of the index type and a value of the component type.

```
type Color is (Red, Green, Blue);
type Intensity is range 0 .. 255;

type Colored_Point is array (Color) of Intensity;
```

The type `Color` is the index type and the type `Intensity` is the component type of the array type `Colored_Point`. See array[42].

5.11 See also

See also

5.11.1 Wikibook

- Ada Programming[43]

5.11.2 Ada Reference Manual

- 3.2.1 Type Declarations ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-2-1.html}
- 3.3 Objects and Named Numbers ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-3.html}
- 3.7 Discriminants ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-7.html}
- 3.10 Access Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-10.html}
- 4.9 Static Expressions and Static Subtypes ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-9.html}
- 13.9 Unchecked Type Conversions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-13-9.html}

42 Chapter 11 on page 83
43 http://en.wikibooks.org/wiki/Ada%20Programming

- 13.3 Operational and Representation Attributes ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-13-3.html}
- Annex K (informative) Language-Defined Attributes ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-K.html}

6 Integer types

A **range** is a signed integer value which ranges from a *First* to a last *Last*. It is defined as

```
range First .. Last
```

When a value is assigned to an object with such a range constraint, the value is checked for validity and Constraint_Error exception[1] is raised when the value is not within *First* to *Last*.

When declaring a range type, the corresponding mathematical operators are implicitly declared by the language at the same place.

The compiler is free to choose a suitable underlaying hardware type for this user defined type.

6.1 Working demo

Working demo

The following Demo defines a new range from -5 to 10 and then prints the whole range out.

File: range_1.adb

```
with Ada ;

procedure Range_1 is
   type Range_Type is range -5 .. 10;

   package T_IO renames Ada.Text_IO;
   package I_IO is new Ada.Text_IO.Integer_IO (Range_Type);

begin
   for A in Range_Type loop
      I_IO.Put (
         Item  => A,
         Width => 3,
         Base  => 10);

      if A < Range_Type'Last then
         T_IO.Put (",");
      else
         T_IO.New_Line;
```

1 Chapter 19 on page 153

```
      end if;
   end loop;
end Range_1;
```

6.2 See also

See also

6.2.1 Wikibook

- Ada Programming[2]
- Ada Programming/Types[3]
- Ada Programming/Keywords/range[4]

6.2.2 Ada Reference Manual

- 4.4 Expressions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-4.html}
- 3.5.4 Integer Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-4.html}

es:Programación en Ada/Tipos/Enteros[5]

2 http://en.wikibooks.org/wiki/Ada%20Programming
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Frange
5 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FTipos%2FEnteros

7 Unsigned integer types

7.1 Description

Description

Unsigned integers in Ada have a value range from 0 to some positive number (not necessarily 1 subtracted from some power of 2). They are defined using the **mod** keyword because they implement a wrap-around arithmetic.

```
mod Modulus
```

where '*First* is 0 and '*Last* is Modulus - 1.

Wrap-around arithmetic means that 'Last + 1 = 0 = 'First, and 'First - 1 = 'Last. Additionally to the normal arithmetic operators, bitwise **and**, **or** and **xor** are defined for the type.

The predefined package Interfaces (RM B.2 ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-2.html}) presents unsigned integers based on powers of 2

```
type Unsigned_n is mod 2**n;
```

for which also shift and rotate operations are defined. The values of n depend on compiler and target architecture.

You can use **range** to sub-range a modular type:

```
type Byte is mod 256;
subtype Half_Byte is Byte range 0 .. 127;
```

But beware: the Modulus of Half_Byte is still 256! Arithmetic with such a type is interesting to say the least.

7.2 See also

See also

7.2.1 Wikibook

- Ada Programming[1]
- Ada Programming/Types[2]
- Ada Programming/Keywords/mod[3]

7.2.2 Ada Reference Manual

- 4.4 Expressions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-4.html}
- 3.5.4 Integer Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-4.html}

1 http://en.wikibooks.org/wiki/Ada%20Programming
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fmod

8 Enumerations

An **enumeration** type is defined as a list of possible values:

```
type Primary_Color is (Red, Green, Blue);
```

Like for numeric types, where e.g. 1 is an integer literal, Red, Green and Blue are called the literals of this type. There are no other values assignable to objects of this type.

8.1 Operators and attributes

Operators and attributes

Apart from equality ("="), the only operators on enumeration types are the ordering operators[1]: "<", "<=", "=", "/=", ">=", ">", where the order relation is given implicitly by the sequence of literals: Each literal has a position, starting with 0 for the first, incremented by one for each successor. This position can be queried via the 'Pos attribute[2]; the inverse is 'Val, which returns the corresponding literal. In our example:

```
Primary_Color'Pos (Red) = 0
Primary_Color'Val (0)   = Red
```

There are two other important attributes: *Image* and *Value* (don't confuse *Val* with *Value*). *Image* returns the string representation of the value (in capital letters), *Value* is the inverse:

```
Primary_Color'Image ( Red ) = "RED"
Primary_Color'Value ("Red") =  Red
```

These attributes are important for simple IO[3] (there are more elaborate IO facilities in Ada for enumeration types). Note that, since Ada is case-insensitive, the string given to 'Value can be in any case.

8.2 Enumeration literals

1 Chapter 37 on page 301
2 Chapter 38 on page 305
3 Chapter 18 on page 147

Enumeration literals

Literals are overloadable, i.e. you can have another type with the same literals.

```
type Traffic_Light is (Red, Yellow, Green);
```

Overload resolution within the context of use of a literal normally resolves which Red is meant. Only if you have an unresolvable overloading conflict, you can qualify with special syntax which Red is meant:

```
Primary_Color'(Red)
```

Like many other declarative items, enumeration literals can be renamed. In fact, such a literal is a actually function[4], so it has to be renamed as such:

```
function Red return P.Primary_Color renames P.Red;
```

Here, Primary_Color is assumed to be defined in package P, which is visible at the place of the renaming declaration. Renaming makes Red directly visible without necessity to resort the use-clause.

Note that redeclaration as a function does not affect the staticness of the literal.

8.2.1 Characters as enumeration literals

Rather unique to Ada is the use of character literals as enumeration literals:

```
type ABC is ('A', 'B', 'C');
```

This literal 'A' has **nothing** in common with the literal 'A' of the predefined type Character (or Wide_Character).

Every type that has at least one character literal is a character type. For every character type, string literals and the concatenation operator "&"[5] are also implicitly defined.

```
type My_Character is (No_Character, 'a', Literal, 'z');
type My_String is array (Positive range <>) of My_Character;

S: My_String := "aa" & Literal & "za" & 'z';
T: My_String := ('a', 'a', Literal, 'z', 'a', 'z');
```

In this example, S and T have the same value.

4 Chapter 16.2 on page 127
5 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%26

Ada's Character type is defined that way. See Ada Programming/Libraries/Standard[6].

8.2.2 Booleans as enumeration literals

Also Booleans are defined as enumeration types:

```
type Boolean is (False, True);
```

There is special semantics implied with this declaration in that objects and expressions of this type can be used as conditions. Note that the literals False and True are not Ada keywords.

Thus it is not sufficient to declare a type with these literals and then hope objects of this type can be used like so:

```
type My_Boolean is (False, True);
Condition: My_Boolean;

if Condition then  -- wrong, won't compile
```

If you need your own Booleans (perhaps with special size requirements), you have to derive from the predefined Boolean:

```
type My_Boolean is new Boolean;
Condition: My_Boolean;

if Condition then  -- OK
```

8.3 Enumeration subtypes

Enumeration subtypes

You can use **range** to subtype an enumeration type:

```
subtype Capital_Letter is Character range 'A' .. 'Z';
```

```
type Day_Of_Week is (Sunday, Monday, Tuesday, Wednesday, Thursday,
Friday, Saturday);

subtype Working_Day is Day_Of_Week range Monday .. Friday;
```

8.4 See also

6 Chapter 41 on page 333

See also

8.4.1 Wikibook

- Ada Programming[7]
- Ada Programming/Types[8]
- Ada Programming/Libraries/Standard[9]

8.4.2 Ada Reference Manual

- 3.5.1 Enumeration Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-1.html}

es:Programación en Ada/Tipos/Enumeraciones[10]

7 http://en.wikibooks.org/wiki/Ada%20Programming
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes
9 Chapter 41 on page 333
10 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FTipos%2FEnumeraciones

9 Floating point types

9.1 Description

Description

w:Floating point[1]

To define a floating point type, you only have to say how many **digits**[2] are needed, i.e. you define the relative precision:

```
digits Num_Digits
```

If you like, you can declare the minimum range needed as well:

```
digits Num_Digits range[3] Low .. High
```

This facility is a great benefit of Ada over (most) other programming languages. In other languages, you just choose between "float" and "long float", and what most people do is:

- choose float if they don't care about accuracy
- otherwise, choose long float, because it is the best you can get

In either case, you don't know what accuracy you get.

In Ada, you specify the accuracy you need, and the compiler will choose an appropriate floating point type with *at least* the accuracy you asked for. This way, your requirement is guaranteed. Moreover, if the computer has more than two floating point types available, the compiler can make use of all of them.

9.2 See also

See also

9.2.1 Wikibook

- Ada Programming[4]

1 http://en.wikipedia.org/wiki/Floating%20point
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fdigits
4 http://en.wikibooks.org/wiki/Ada%20Programming

- Ada Programming/Types[5]
- Ada Programming/Types/range[6]
- Ada Programming/Types/delta[7]
- Ada Programming/Types/mod[8]
- Ada Programming/Keywords/digits[9]

9.2.2 Ada Reference Manual

- 3.5.7 Floating Point Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-7.html}

es:Programación en Ada/Tipos/Coma flotante[10]

5 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes
6 Chapter 6 on page 69
7 Chapter 10 on page 79
8 Chapter 7 on page 71
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fdigits
10 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FTipos%2FComa%20flotante

10 Fixed point types

10.1 Description

Description

w:Fixed-point arithmetic[1] A fixed point type defines a set of values that are evenly spaced with a given absolute precision. In contrast, floating point values are all spaced according to a relative precision.

The absolute precision is given as the delta of the type. There are two kinds of fixed point types, ordinary and decimal.

For **Ordinary Fixed Point** types, the delta gives a hint to the compiler how to choose the small value if it is not specified: It can be *any integer power of two* not greater than delta. You may specify the small via an attribute clause to be *any value* not greater than delta. (If the compiler cannot conform to this small value, it has to reject the declaration.)

For **Decimal Fixed Point** types, the small is defined to be the delta, which in turn must be an integer power of ten. (Thus you cannot specify the small by an attribute clause.)

For example, if you define a decimal fixed point type with a delta of 0.1, you will be able to accurately store the values 0.1, 1.0, 2.2, 5.7, etc. You will not be able to accurately store the value 0.01. Instead, the value will be rounded down to 0.0.

If the compiler accepts your fixed point type definition, it guarantees that values represented by that type will have at least the degree of accuracy specified (or better). If the compiler cannot support the type definition (e.g. due to limited hardware) then a compile-time error will result.

10.2 Ordinary Fixed Point

Ordinary Fixed Point

For an ordinary fixed point, you just define the delta and a range:

```
delta Delta range Low .. High
```

The delta can be any real value — for example you may define a circle with one arcsecond resolution with:

1 http://en.wikipedia.org/wiki/Fixed-point%20arithmetic

```
delta 1 / (60 * 60) range 0.0 .. 360.0
```

[There is one rather strange rule about fixed point types: Because of the way they are internally represented, the range might only go up to `'Last - Delta`. This is a bit like a circle — the 0° and 360° mark is also the same.]

It should be noted that in the example above the smallest possible value used is not $\frac{1}{60^2} = \frac{1}{3600}$. The compiler will choose a smaller value which, by default, is an integer power of 2 not greater than the delta. In our example this could be $2^{-12} = \frac{1}{4096}$. In most cases this should render better performance but sacrifices precision for it.

If this is not what you wish and precision is indeed more important, you can choose your own small value via the attribute clause '*Small*.

```
type Angle is delta Pi/2.0**31 range -Pi .. Pi;
for Angle'Small use Pi/2.0**31;
```

As internal representation, you will get a 32 bit signed integer type.

10.3 Decimal Fixed Point

Decimal Fixed Point

You define a decimal fixed point by defining the delta and the number of digits needed:

```
delta Delta digits Num_Digits
```

Delta must be a positive or negative integer power of 10 — otherwise the declaration is illegal.

```
delta 10.0**(+2) digits 12
delta 10.0**(-2) digits 12
```

If you like, you can also define the range needed:

```
delta Delta_Value digits Num_Digits range Low .. High
```

10.4 Differences between Ordinary and Decimal Fixed Point Types

Differences between Ordinary and Decimal Fixed Point Types

There is an alternative way of declaring a "decimal" fixed point: You declare an ordinary fixed point and use an integer power of 10 as '*Small*. The following two declarations are

equivalent with respect to the internal representation:

```
-- decimal fixed point

type Duration is delta 10.0**(-9) digits 9;
```

```
-- ordinary fixed point

type Duration is delta 10.0**(-9) range -1.0 .. 1.0;
for Duration'Small use 10.0**(-9);
```

You might wonder what the difference then is between these two declarations. The answer is:

None with respect to precision, addition, subtraction, multiplication with integer values.

The following is an incomplete list of differences between ordinary and decimal fixed point types.

- Decimal fixed point types are intended to reflect typical **COBOL** declarations with a given number of digits.
- Truncation is required for decimal, not for ordinary, fixed point in multiplication and division (RM 4.5.5 (21) ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-4-5-5.html}) and type conversions. Operations on decimal fixed point are fully specified, which is not true for ordinary fixed point.
- The following attributes are only defined for decimal fixed point: T'Digits (RM 3.5.10 (10) ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-5-10.html}) corresponds to the number of decimal digits that are representable; T'Scale (RM 3.5.10 (11) ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-5-10.html} , taken from **COBOL**) indicates the position of the point relative to the rightmost significant digits; T'Round (RM 3.5.10 (12) ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-5-10.html}) can be used to specify rounding on conversion.
- Package Decimal (RM F.2 ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-F-2.html}), which of course applies only to decimal fixed point, defines the decimal Divide generic procedure. If annex F is supported (GNAT does), at least 18 digits must be supported (there is no such rule for fixed point).
- Decimal_IO (RM A.10.1 (73) ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-A-10-1.html}) has semantics different from Fixed_IO (RM A.10.1 (68) ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-A-10-1.html}).
- Static expressions must be a multiple of the Small for decimal fixed point.

Conclusion: For normal numeric use, an ordinary fixed point (probably with 'Small defined) should be defined. Only if you are interested in COBOL like use, i.e. well defined deterministic decimal semantics (especially for financial computations, but that might apply to cases other than money) should you take decimal fixed point.

10.5 See also

See also

10.5.1 Wikibook

- Ada Programming[2]
- Ada Programming/Types[3]
- Ada Programming/Types/range[4]
- Ada Programming/Types/digits[5]
- Ada Programming/Types/mod[6]
- Ada Programming/Keywords/delta[7]
- Ada Programming/Attributes/'Small[8]

10.5.2 Ada 95 Reference Manual

- 3.5.9 Fixed Point Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-5-9.html}

10.5.3 Ada 2005 Reference Manual

- 3.5.9 Fixed Point Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-9.html}

es:Programación en Ada/Tipos/Coma fija[9]

2 http://en.wikibooks.org/wiki/Ada%20Programming
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes
4 Chapter 6 on page 69
5 Chapter 9 on page 77
6 Chapter 7 on page 71
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fdelta
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Small
9 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FTipos%2FComa%20fija

11 Arrays

An array[1] is a collection of elements which can be accessed by one or more index values. In Ada any definite type is allowed as element and any discrete type, i.e. Range[2], Modular[3] or Enumeration[4], can be used as an index.

11.1 Declaring arrays

Declaring arrays

Ada's arrays are quite powerful and so there are quite a few syntax variations, which are presented below.

11.1.1 Basic syntax

The basic form of an Ada array is:

```
array (Index_Range) of Element_Type
```

where Index_Range is a range of values within a discrete index type, and Element_Type is a definite subtype. The array consists of one element of "Element_Type" for each possible value in the given range. If you for example want to count how often a specific letter appears inside a text, you could use:

```
type Character_Counter is array (Character) of Natural;
```

As a general advice, do not use Integer as the index range, since most of the time negative indices do not make sense. It is also a good style when using numeric indices, to define them starting in 1 instead of 0, since it is more intuitive for humans and avoids off-by-one error[5]s.

1 http://en.wikipedia.org/wiki/array
2 Chapter 6 on page 69
3 Chapter 7 on page 71
4 Chapter 8 on page 73
5 http://en.wikipedia.org/wiki/off-by-one%20error

11.1.2 With known subrange

Often you don't need an array of all possible values of the index type. In this case you can **subtype** your index type to the actually needed range.

```
subtype Index_Sub_Type is Index_Type range First .. Last

array (Index_Sub_Type) of Element_Type
```

Since this may involve a lot of typing and you may also run out of useful names for new subtypes[6], the array declaration allows for a shortcut:

```
array (Index_Type range First .. Last) of Element_Type
```

Since `First` and `Last` are expressions of `Index_Type`, a simpler form of the above is:

```
array (First .. Last) of Element_Type
```

Note that if `First` and `Last` are numeric literals, this implies the index type `Integer`.

If in the example above the character counter should only count upper case characters and discard all other characters, you can use the following array type:

```
type Character_Counter is array (Character range 'A' .. 'Z') of Natural;
```

11.1.3 With unknown subrange

Sometimes the range actually needed is not known until runtime or you need objects of different lengths. In some languages you would resort to pointers to element types. Not with Ada. Here we have the box '<>', which allows us to declare indefinite arrays:

```
array (Index_Type range <>) of Element_Type;
```

When you declare objects of such a type, the bounds must of course be given and the object is constrained to them.

The predefined type String[7] is such a type. It is defined as

```
type String is array (Positive range <>) of Character;
```

6 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes

7 Chapter 15 on page 119

You define objects of such an unconstrained type in several ways (the extrapolation to other arrays than String should be obvious):

```
Text : String (10 .. 20);
Input: String := Read_from_some_file;
```

(These declarations additionally define anonymous subtypes of String.) In the first example, the range of indices is explicitly given. In the second example, the range is implicitly defined from the initial expression, which here could be via a function reading data from some file. Both objects are constrained to their ranges, i.e. they cannot grow nor shrink.

11.1.4 With aliased elements

If you come from C[8]/C++[9], you are probably used to the fact that every element of an array has an address. The C[10]/C++[11] standards actually demand that.

In Ada, this is not true. Consider the following array:

```
type Day_Of_Month is range 1 .. 31;
type Day_Has_Appointment is array (Day_Of_Month) of Boolean;
pragma  (Day_Has_Appointment);
```

Since we have packed the array, the compiler will use as little storage as possible. And in most cases this will mean that 8 boolean values will fit into one byte.

So Ada knows about arrays where more than one element shares one address. So what if you need to address each single element. Just not using **pragma**[12] *Pack*[13] is not enough. If the CPU[14] has very fast bit access, the compiler might pack the array without being told. You need to tell the compiler that you need to address each element via an access.

```
type Day_Of_Month is range 1 .. 31;
type Day_Has_Appointment is array (Day_Of_Month) of aliased Boolean;
```

11.1.5 Arrays with more than one dimension

Arrays can have more than one index. Consider the following 2-dimensional array:

8 http://en.wikibooks.org/wiki/Programming%3AC
9 http://en.wikibooks.org/wiki/Programming%3AC_plus_plus
10 http://en.wikibooks.org/wiki/Programming%3AC
11 http://en.wikibooks.org/wiki/Programming%3AC_plus_plus
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fpragma
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPack
14 http://en.wikipedia.org/wiki/CPU

```
type Character_Display is
   array ( Positive range <> ,  Positive range <> )  of Character;
```

This type permits declaring rectangular arrays of characters. Example:

```
Magic_Square:   constant Character_Display :=
   ( ( ' S' ,  ' A' ,  ' T' ,  ' O' ,  ' R' ) ,
    ( ' A' ,  ' R' ,  ' E' ,  ' P' ,  ' O' ) ,
    ( ' T' ,  ' E' ,  ' N' ,  ' E' ,  ' T' ) ,
    ( ' O' ,  ' P' ,  ' E' ,  ' R' ,  ' A' ) ,
    ( ' R' ,  ' O' ,  ' T' ,  ' A' ,  ' S' ) ) ;
```

Or, stating some index values explicitly,

```
Magic_Square:   constant Character_Display( 1 ..   5,  1 ..   5)  :=
   ( 1 =>  ( ' S' ,  ' A' ,  ' T' ,  ' O' ,  ' R' ) ,
    2 =>  ( ' A' ,  ' R' ,  ' E' ,  ' P' ,  ' O' ) ,
    3 =>  ( ' T' ,  ' E' ,  ' N' ,  ' E' ,  ' T' ) ,
    4 =>  ( ' O' ,  ' P' ,  ' E' ,  ' R' ,  ' A' ) ,
    5 =>  ( ' R' ,  ' O' ,  ' T' ,  ' A' ,  ' S' ) ) ;
```

The index values of the second dimension, those indexing the characters in each row, are in 1 .. 5 here. By choosing a different second range, we could change these to be in 11 .. 15:

```
Magic_Square:   constant Character_Display( 1 ..   5,  11 ..   15)  :=
   ( 1 =>  ( ' S' ,  ' A' ,  ' T' ,  ' O' ,  ' R' ) ,
     ...
```

By adding more dimensions to an array type, we could have squares, cubes (or « bricks »), etc., of homogenous data items.

Finally, an array of characters is a string (see Ada Programming/Strings[15]). Therefore, `Magic_Square` may simply be declared like this:

```
Magic_Square:   constant Character_Display :=
   ( "SATOR",
    "AREPO",
    "TENET",
    "OPERA",
    "ROTAS") ;
```

11.2 Using arrays

15 Chapter 15 on page 119

11.2.1 Assignment

When accessing elements, the index is specified in parentheses. It is also possible to access slices in this way:

```
Vector_A (1 .. 3) := Vector_B (3 .. 5);
```

Note that the index range slides in this example: After the assignment, Vector_A (1) = Vector_B (3) and similarly for the other indices.

Also note that the ranges overlap, nevertheless Vector_A (3) /= Vector_B (3); a compiler delivering such a result would be severely broken.

11.2.2 Concatenate

The operator "&" can be used to concatenate arrays:

```
Name := First_Name & ' ' & Last_Name;
```

In both cases, if the resulting array does not fit in the destination array, Constraint_Error is raised.

If you try to access an existing element by indexing outside the array bounds, Constraint_-Error is raised (unless checks are suppressed).

11.2.3 Array Attributes

There are four Attributes which are important for arrays: '*First*, '*Last*, '*Length* and '*Range*. Lets look at them with an example. Say we have the following three strings:

```
Hello_World  : constant String := "Hello World!";
World        : constant String := Hello_World (7 .. 11);
Empty_String : constant String := "";
```

Then the four attributes will have the following values:

Array	**'*First***	**'*Last***	**'*Length***	**'*Range***
Hello_World	1	12	12	1 .. 12
World	7	11	5	7 .. 11
Empty_String	1	0	0	1 .. 0

The example was chosen to show a few common beginner's mistakes:

1. The assumption that strings begin with the index value 1 is wrong.

2. The assumption (which follows from the first one) that X'Length = X'Last is wrong.
3. And last the assumption that X'Last >= X'First; this is not true for empty strings.

The attribute '*Range* is a little special as it does not return a discrete value but an abstract description of the array. One might wonder what it is good for. The most common use is in the for loop on arrays[16] but '*Range* can also be used in declaring a name for the index subtype:

```
subtype Hello_World_Index is Integer range Hello_World'Range;
```

The *Range* attribute can be convenient when programming index checks:

```
if K in World' Range then
   return World( K) ;
else
   return Substitute;
end if;
```

11.2.4 Empty or Null Arrays

As you have seen in the section above, Ada allows for empty arrays. And — of course — you can have empty arrays of all sorts, not just String:

```
type Some_Array is array (Positive range <>) of Boolean;

Empty_Some_Array : constant Some_Array (1 .. 0) := (others => False);
```

Note: If you give an initial expression to an empty array (which is a must for a constant), the expression in the aggregate will of course not be evaluated since there are no elements actually stored.

11.3 See also

See also

11.3.1 Wikibook

- Ada Programming[17]
- Ada Programming/Types[18]
- Data Structures[19]
- Data Structures/Arrays[20]

16 Chapter 4 on page 37
17 http://en.wikibooks.org/wiki/Ada%20Programming
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes
19 http://en.wikibooks.org/wiki/Data%20Structures
20 http://en.wikibooks.org/wiki/Data%20Structures%2FArrays

11.3.2 Ada 95 Reference Manual

- 3.6 Array Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-6.html}

11.3.3 Ada 2005 Reference Manual

- 3.6 Array Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-6.html}

11.3.4 Ada Quality and Style Guide

- 10.5.7 Packed Boolean Array Shifts ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_10/10-5-7.html}

es:Programación en Ada/Tipos/Arrays[21]

21 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FTipos%2FArrays

12 Records

A **record** is a composite type[1] that groups one or more fields. A field can be of any type, even a record.

12.1 Basic record

Basic record

```
type Basic_Record is
   record
      A : Integer;
   end record;
```

12.2 Null record

Null record

The null record is when a type without data is needed. There are two ways to declare a null record:

```
type Null_Record is
   record
      null;
   end record;
```

```
type Null_Record is null record;
```

For the compiler they are the same. However, programmers often use the first variant if the type is not finished yet to show that they are planning to expand the type later, or they usually use the second if the (tagged) record is a base class in object oriented programming.

12.3 Record Values

Record Values

Values of a record type can be specified using a record aggregate, giving a list of named components thus

1 http://en.wikibooks.org/wiki/ada%20Programming%2FTypes%23List%20of%20types

```
   A_Basic_Record       :   Basic_Record         :=  Basic_Record' ( A =>
42) ;
   Another_Basic_Record :   Basic_Record         :=  ( A =>  42) ;
   Nix                  :   constant Null_Record :=  ( null record) ;
```

Given a somewhat larger record type,

```
   type Car is record
      Identity       :   Long_Long_Integer;
      Number_Wheels  :   Positive range 1 ..   10;
      Paint          :   Color;
      Horse_Power_kW :   Float range 0.  0 ..   2_000.  0;
      Consumption    :   Float range 0.  0 ..   100.  0;
   end record;
```

a value *may* be specified using *positional* notation, that is, specifying a value for each record component in declaration order

```
   BMW :   Car :=  ( 2007_752_83992434,  5,  Blue,  190.  0,  10.  1) ;
```

However, naming the components of a `Car` aggregate offers a number of advantages.

1. Easy identification of which value is used for which component. (After all, named components are the raison d'être of records.)
2. Reordering the components is allowed—you only have to remember the component names, not their position.
3. Improved compiler diagnostic messages.

Reordering components is possible because component names will inform the compiler (and the human reader!) of the intended value associations. Improved compiler messages are also in consequence of this additional information passed to the compiler. While an omitted component will always be reported due to Ada's coverage rules[2], messages can be much more specific when there are named associations. Considering the `Car` type from above, suppose a programmer by mistake specifies only one of the two floating point values for `BMW` in positional notation. The compiler, in search of another component value, will then not be able to decide whether the specified value is intended for `Horse_Power_kW` or for `Consumption`. If the programmer instead uses named association, say Horse_Power_kW => 190. 0, it will be clear which other component is missing.

```
   BMW :   Car :=
     ( Identity       =>  2007_752_83992434,
       Number_Wheels  =>  5,
       Horse_Power_kW =>  190.  0,
       Consumption    =>  10.  1,
       Paint          =>  Blue) ;
```

In order to access a component of a record instance, use the dot delimiter (.), as in `BMW.Number_Wheels`.

2 http://www.adacore.com/2007/05/14/gem-1/

12.4 Discriminated record

Discriminated record

```
type Discriminated_Record (Size : Natural) is
   record
      A : String (1 .. Size);
   end record;
```

12.5 Variant record

Variant record

The variant record is a special type of discriminated record where the presence of some components depend on the value of the discriminant.

```
type Traffic_Light is (Red, Yellow, Green);

type Variant_Record (Option : Traffic_Light) is
   record
      --common components

      case Option is
         when Red =>
            --components for red
         when Yellow =>
            --components for yellow
         when Green =>
            --components for green
      end case;
   end record;
```

12.5.1 Mutable and immutable variant records

You can declare variant record types such that its discriminant, and thus its variant structure, can be changed during the lifetime of the variable. Such a record is said to be *mutable*. When "mutating" a record, you must assign **all** components of the variant structure which you are mutating at once, replacing the record with a complete variant structure. Although a variant record declaration may allow objects of its type to be mutable, there are certain restrictions on whether the objects will be mutable. Reasons restricting an object from being mutable include:

- the object is declared with a discriminant (see Immutable_Traffic_Light below)
- the object is aliased (either by use of **aliased** in the object declaration, or by allocation on the heap using **new**)

```
  type Traffic_Light is (Red, Yellow, Green);

  type Mutable_Variant_Record (Option : Traffic_Light := Red) is      --the discriminant must
have a default value
     record
        --common components
        Location : Natural;
```

```
        case Option is
           when Red =>
              --components for red
              Flashing : Boolean := True;
           when Yellow =>
              --components for yellow
              Timeout    : Duration := 0.0;
           when Green =>
              --components for green
              Whatever : Positive := 1;
        end case;
     end record;
...
Mutable_Traffic_Light   : Mutable_Variant_Record;
--not declaring a discriminant makes this record mutable

--it has the default discriminant/variant

--structure and values

Immutable_Traffic_Light : Mutable_Variant_Record (Option => Yellow);
--this record is immutable, the discriminant cannot be changed

--even though the type declaration allows for mutable objects

--with different discriminant values
...
Mutable_Traffic_Light   := (Option => Yellow,
--mutation requires assignment of all components
                            Location => 54,
--for the given variant structure
                            Timeout => 2.3);
...
--restrictions on objects, causing them to be immutable
type Traffic_Light_Access is access Mutable_Variant_Record;
Any_Traffic_Light       : Traffic_Light_Access :=
                             new Mutable_Variant_Record;
Aliased_Traffic_Light   : aliased Mutable_Variant_Record;
```

Conversely, you can declare record types so that the discriminant along with the structure of the variant record may not be changed. To make a record type declaration *immutable*, the discriminant must **not** have a default value.

```
  type Traffic_Light is (Red, Yellow, Green);

  type Immutable_Variant_Record (Option : Traffic_Light) is --no default value makes the record
type immutable
     record
        --common components
        Location : Natural := 0;
        case Option is
           when Red =>
              --components for red
              Flashing : Boolean := True;
           when Yellow =>
              --components for yellow
              Timeout    : Duration;
           when Green =>
              --components for green
              Whatever : Positive := 1;
        end case;
     end record;
```

```
...
Default_Traffic_Light   : Immutable_Variant_Record;
  --ILLEGAL!
Immutable_Traffic_Light : Immutable_Variant_Record (Option =>
Yellow); --this record is immutable, since the type declaration is immutable
```

12.6 Union

Union

This language feature is only available in Ada 2005

```
type Traffic_Light is (Red, Yellow, Green);

type Union (Option : Traffic_Light := Traffic_Light'First) is
   record
      --common components

      case Option is
         when Red =>
            --components for red
         when Yellow =>
            --components for yellow
         when Green =>
            --components for green
      end case;
   end record;

pragma  (Union);
pragma  (C, Union);    --optional
```

The difference to a variant record is such that *Option* is not actually stored inside the record and never checked for correctness - it's just a dummy.

This kind of record is usually used for interfacing with C but can be used for other purposes as well (then without `pragma  (C, Union);`).

12.7 Tagged record

Tagged record

The tagged record is one part of what in other languages is called a class. It is the basic foundation of object orientated programming in Ada[3]. The other two parts a class in Ada needs is a package[4] and primitive operations[5].

3 Chapter 22 on page 187
4 Chapter 22.4.2 on page 217
5 Chapter 22.1.2 on page 188

```
type Person is tagged
   record
      Name   : String (1 .. 10);
      Gender : Gender_Type;
   end record;
```

```
type Programmer is new Person with
   record
      Skilled_In : Language_List;
   end record;
```

Ada 2005 only:

```
type Programmer is new Person
                   and Printable
with
   record
      Skilled_In : Language_List;
   end record;
```

12.8 Abstract tagged record

Abstract tagged record

An abstract type has at least an abstract primitive operation, i.e. one of its operations is not defined and then its derivative types has to provide an implementation.

12.9 With aliased elements

With aliased elements

If you come from C[6]/C++[7], you are probably used to the fact that every element of a record - which is not part of a bitset - has an address. In Ada, this is not true because records, just like arrays, can be packed. And just like arrays you can use **aliased** to ensure that an element can be accessed via an access type.

```
type Basic_Record is
   record
      A : aliased Integer;
   end record ;
```

Please note: each element needs its own **aliased**.

6 http://en.wikibooks.org/wiki/C%20Programming
7 http://en.wikibooks.org/wiki/C%2B%2B%20Programming

12.10 Limited Records

Limited Records

In addition to being variant, tagged, and abstract, records may also be limited (no assignment, and no predefined equality operation for Limited Types[8]). In object oriented programming, when tagged objects are handled by references instead of copying them, this blends well with making objects limited.

12.11 See also

See also

12.11.1 Wikibook

- Ada Programming[9]
- Ada Programming/Types[10]
- Ada Programming/Keywords/record[11]
- Ada Programming/Keywords/null[12]
- Ada Programming/Keywords/abstract[13]
- Ada Programming/Keywords/case[14]
- Ada Programming/Keywords/when[15]
- Ada Programming/Pragmas/Unchecked_Union[16]

12.11.2 Ada Reference Manual

Ada 95

- 3.8 Record Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-8.html}

Ada 2005

- 3.8 Record Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-8.html}
- Annex B.3.3 Pragma Unchecked_Union ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-3-3.html}

8 Chapter 14 on page 113
9 http://en.wikibooks.org/wiki/Ada%20Programming
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Frecord
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fnull
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fabstract
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fcase
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fwhen
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnchecked_Union

Ada Issues

- AI95-00216-01 Unchecked unions — variant records with no run-time discriminant ^{`http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00216.TXT`}

13 Access types

13.1 What's an Access Type?

What's an Access Type?

Access types in Ada are what other languages call pointers. They point to objects located at certain addresses. So normally one can think of access types as simple addresses (there are exceptions from this simplified view). Ada instead of saying *points to* talks of *granting access to* or *designating* an object.

Objects of access types are implicitly initialized with **null**, i.e. they point to nothing when not explicitly initialized.

Access types should be used rarely in Ada. In a lot of circumstances where pointers are used in other languages, there are other ways without pointers. If you need dynamic data structures, first check whether you can use the Ada Container library. Especially for indefinite record or array components, the Ada 2012 package Indefinite_Holders (RM A.18.18) can be used instead of pointers.

There are four kinds of access types in Ada: Pool access types - General access types - Anonymous access types - Access to subprogram types.

13.2 Pool access

Pool access

A *pool access type* handles accesses to objects which were created on some specific heap (or storage pool as it is called in Ada). A pointer of these types cannot point to a stack or library level (static) object or an object in a different storage pool. Therefore, conversion between pool access types is illegal. (Unchecked_Conversion may be used, but note that deallocation via an access object with a storage pool different from the one it was allocated with is erroneous.)

```
type Person is record
  First_Name : String (1..30);
  Last_Name  : String (1..20);
end record;

type Person_Access is access Person;
```

A size clause may be used to limit the corresponding (implementation defined anonymous) storage pool. A size clause of 0 disables calls of an allocator.

```
for Person_Access'Size use 0;
```

The storage pool is implementation defined if not specified. Ada supports user defined storage pools, so you can define the storage pool with

```
for Person_Access'Storage_Pool use Pool_Name;
```

Objects in a storage pool are created with the keyword **new**:

```
Father: Person_Access := new Person;
       -- uninitialized
Mother: Person_Access := new Person'(Mothers_First_Name,
Mothers_Last_Name);  -- initialized
```

You access the object in the storage pool by appending `.all`. `Mother.all` is the complete record; components are denoted as usual with the dot notation: `Mother.all.First_Name`. When accessing components, *implicit dereferencing* (i.e. omitting **all**) can serve as a convenient shorthand:

```
Mother.all := (Last_Name => Father.Last_Name, First_Name =>
Mother.First_Name);  -- marriage
```

Implicit dereferencing also applies to arrays:

```
  type Vector is array (1 .. 3) of Complex;
  type Vector_Access is access Vector;

  VA: Vector_Access := new Vector;
  VB: array (1 .. 3) of Vector_Access := (others => new Vector);

  C1: Complex := VA (3);    -- a shorter equivalent for VA   .all (3)
  C2: Complex := VB (3)(1); -- a shorter equivalent for VB(3).all (1)
```

Be careful to discriminate between deep and shallow copies when copying with access objects:

```
Obj1.all := Obj2.all;  -- Deep copy: Obj1 still refers to an object
                       -- different from Obj2, but it has the same
content
Obj1 := Obj2;          -- Shallow copy: Obj1 now refers to the same
object as Obj2
```

13.2.1 Deleting objects from a storage pool

Although the Ada standard mentions a garbage collector, which would automatically remove all unneeded objects that have been created on the heap (when no storage pool has been defined), only Ada compilers targeting a virtual machine like Java or .NET actually have garbage collectors. ~~There is also a **pragma Controlled**, which, when applied to such an access type, prevents automatic garbage collection of objects created with it.~~ Note that **pragma** `Controlled` will be dropped from Ada 2012, see RM 2012 13.11.3.

Therefore in order to delete an object from the heap, you need the generic unit Ada . Apply utmost care to not create dangling pointers when deallocating objects as is shown in the example below. (And note that deallocating objects with a different access type than the one with which they were created is erroneous when the corresponding storage pools are different.)

```
with Ada ;

procedure Deallocation_Sample is

   type Vector     is array (Integer range <>) of Float;
   type Vector_Ref is access Vector;

   procedure Free_Vector is new Ada
      (Object => Vector, Name => Vector_Ref);

   VA, VB: Vector_Ref;
   V     : Vector;

begin

   VA     := new Vector (1 .. 10);
   VB     := VA;  -- points to the same location as VA

   VA.all := (others => 0.0);

   --  ... Do whatever you need to do with the vector

   Free_Vector (VA); -- The memory is deallocated and VA is now null

   V := VB.all;  -- VB is not null, access to a dangling pointer is
erroneous

end Deallocation_Sample;
```

It is exactly because of this problem with dangling pointers that the deallocation operation is called **unchecked**. It is the chore of the programmer to take care that this does not happen.

Since Ada allows for user defined storage pools, you could also try a garbage collector library[1].

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FAdaCL%23Garbage_Collector

13.2.2 Constructing Reference Counting Pointers

You can find some implementations of reference counting pointers, called *Safe* or *Smart Pointers*, on the net. Using such a type prevents caring about deallocation, since this will automatically be done when there are no more pointers to an object. But be careful - most of those implementations do not prevent deliberate deallocation, thus undermining the alledged safety attained with their use.

A nice tutorial how to construct such a type can be found in a series of Gems on the AdaCore web site.

Gem #97: Reference Counting in Ada – Part 1[2] This little gem constructs a simple reference counted pointer that does not prevent deallocation, i.e. is inherently unsafe.

Gem #107: Preventing Deallocation for Reference-counted Types[3] This further gem describes how to arrive at a pointer type whose safety cannot be compromised (tasking issues aside). The cost of this improved safety is awkward syntax.

Gem #123: Implicit Dereferencing in Ada 2012[4] This gem shows how to simplify the syntax with the new Ada 2012 generation. (Admittedly, this gem is a bit unrelated to reference counting since the new language feature can be applied to any kind of container.)

13.3 General access

General access

General access types grant access to objects created on any storage pool, on the stack or at library level (static). They come in two versions, granting either read-write access or read-only access. Conversions between general access types are allowed, but subject to certain access level checks.

Dereferencing is like for pool access types. Objects (other than pool objects) to be referenced have to be declared **aliased**, and references to them are created with the attribute `'Access`. Access level restrictions prevent accesses to objects from outliving the accessed object, which would make the program erroneous. The attribute `'Unchecked_Access` omits the corresponding checks.

13.3.1 Access to Variable

When the keyword **all** is used in their definition, they grant read-write access.

```
type Day_Of_Month is range 1 .. 31;
type Day_Of_Month_Access is access all Day_Of_Month;
```

2 http://www.adacore.com/2011/01/17/gem-97-reference-counting-in-ada-part-1/
3 http://www.adacore.com/2011/06/06/gem-107-preventing-deallocation-for-reference-counted-types/
4 http://www.adacore.com/adaanswers/gems/gem-123-implicit-dereferencing-in-ada-2012/

13.3.2 Access to Constant

General access types granting read-only access to the referenced object use the keyword **constant** in their definition. The referenced object may be a constant or a variable.

```
type Day_Of_Month is range 1 .. 31;
type Day_Of_Month_Access is access constant Day_Of_Month;
```

13.3.3 Some examples

```
 type General_Pointer  is access all      Integer;
 type Constant_Pointer is access constant Integer;

 I1: aliased constant Integer := 10;
 I2: aliased Integer;

 P1: General_Pointer  := I1'Access;  -- illegal
 P2: Constant_Pointer := I1'Access;  -- OK, read only
 P3: General_Pointer  := I2'Access;  -- OK, read and write
 P4: Constant_Pointer := I2'Access;  -- OK, read only

 P5: constant General_Pointer := I2'Access;  -- read and write only to I2
```

13.4 Anonymous access

Anonymous access

Also *Anonymous access types* come in two versions like general access types, granting either read-write access or read-only access depending on whether the keyword **constant** appears.

An anonymous access can be used as a parameter to a subprogram or as a discriminant. Here are some examples:

```
procedure Modify (Some_Day: access          Day_Of_Month);
procedure Test   (Some_Day: access constant Day_Of_Month);  -- Ada 2005 only
```

```
task type Thread (Execute_For_Day: access Day_Of_Month) is
   ...
end Thread;
```

```
type Day_Data (Store_For_Day: access Day_Of_Month) is record
   --  components
end record;
```

Before using an anonymous access, you should consider a named access type or, even better, consider if the "**out**" or "**in out**" modifier is not more appropriate.

This language feature is only available in Ada 2005

In Ada 2005, anonymous accesses are allowed in more circumstances:

```
type Object is record
  M   : Integer;
  Next: access Object;
end record;

X: access Integer;

function F return access constant Float;
```

13.5 Implicit Dereference

Implicit Dereference

Ada 2012 will simplify accesses to objects via pointers with new syntax.

Imagine you have a container holding some kind of elements.

```
type Container   is private;
type Element_Ptr is access Element;

procedure Put (X: Element; Into: in out Container);
```

Now how do you access elements stored in the container. Of course you can retrieve them by

```
function Get (From: Container) return Element;
```

This will however copy the element, which is unfortunate if the element is big. You get direct access with

```
function Get (From: Container) return Element_Ptr;
```

Now pointers are dangerous since you might easily create dangling pointers like so:

```
P: Element_Ptr := Get (Cont);
P.all := E;
Free (P);
... Get (Cont) -- this is now a dangling pointer
```

Use of an accessor object instead of an access type can prevent inadvertant deallocation (this is still Ada 2005):

```
type Accessor (Data: not null access Element) is limited private;  -- read/write access
function Get (From: Container) return Accessor;
```

(For the null exclusion **not null** in the declaration of the discriminant, see below). Access via such an accessor is safe: The discriminant can only be used for dereferencing, it cannot be copied to an object of type Element_Ptr because its accessibility level is deeper. In the

form above, the accessor provides read and write access. If the keyword **constant** is added, only read access is possible.

```
type Accessor (Data: not null access constant Element) is limited private;  -- only read access
```

Access to the container object now looks like so:

```
Get (Cont).all      := E;  -- via access type: dangerous
Get (Cont).Data.all := E;  -- via accessor: safe, but ugly
```

Here the new Ada 2012 feature of *aspects* comes along handy; for the case at hand, the aspect *Implicit_Dereference* is the one we need:

```
type Accessor (Data: not null access Element) is limited private
   with Implicit_Dereference => Data;
```

Now rather than writing the long and ugly function call of above, we can just omit the discriminant and its dereference like so:

```
Get (Cont).Data.all := E;  -- Ada 2005 via accessor: safe, but ugly
Get (Cont)          := E;  -- Ada 2012 implicit dereference
```

Note that the call `Get (Cont)` is overloaded — it can denote the accessor object or the element, the compiler will select the correct interpretation depending on context.

13.6 Null exclusions

Null exclusions

This language feature is only available in Ada 2005

All access subtypes can be modified with **not null**, objects of such a subtype can then never have the value null, so initializations are compulsory.

```
type    Day_Of_Month_Access          is access   Day_Of_Month;
subtype Day_Of_Month_Not_Null_Access is not null Day_Of_Month_Access;
```

The language also allows to declare *the first subtype* directly with a null exclusion:

```
type Day_Of_Month_Access is not null access Day_Of_Month;
```

However, in nearly all cases this is not a good idea because it renders objects of this type nearly unusable (for example, you are unable to free the allocated memory). Not null accesses are intended for access *subtypes*, object *declarations*, and subprogram *parameters*.`http://groups.google.com/group/comp.lang.ada/msg/13a41ced7af75192`

13.7 Access to Subprogram

Access to Subprogram

An access to subprogram allows to call a subprogram[5] without knowing its name nor its declaration location. One of the uses of this kind of access is the well known callbacks.

```
type Callback_Procedure is access procedure (Id  : Integer;
                                             Text: String);

type Callback_Function is access function (The_Alarm: Alarm) return Natural;
```

For getting an access to a subprogram, the attribute *Access* is applied to a subprogram name with the proper parameter and result profile.

```
procedure Process_Event (Id  : Integer;
                         Text: String);

My_Callback: Callback_Procedure := Process_Event'Access;
```

13.7.1 Anonymous access to Subprogram

This language feature is only available in Ada 2005

```
procedure Test (Call_Back: access procedure (Id: Integer; Text: String));
```

There is now no limit on the number of keyword in a sequence:

```
function F return access function return access function return access Some_Type;
```

This is a function that returns the access to a function that in turn returns an access to a function returning an access to some type.

13.8 Access FAQ

Access FAQ

A few "Frequently Asked Question" and "Frequently Encountered Problems" (mostly from C[6] users) regarding Ada's access types.

5 Chapter 16 on page 125

6 http://en.wikibooks.org/wiki/Programming%3AC

13.8.1 Access vs. access all

An **access all** can do anything a simple **access** can do. So one might ask: "Why use simple **access** at all?" - And indeed some programmers never use simple **access**.

Unchecked_Deallocation is always dangerous if misused. It is just as easy to deallocate a pool-specific object twice, and just as dangerous as deallocating a stack object. The advantage of "access all" is that you may not need to use Unchecked_Deallocation at all.

Moral: if you have (or may have) a valid reason to store an 'Access or 'Unchecked_Access into an access object, then use "access all" and don't worry about it. If not, the mantra of "least privilege" suggests that the "all" should be left out (don't enable capabilities that you are not going to use).

The following (perhaps disastrous) example will try to deallocate a stack object:

```
declare

  type Day_Of_Month is range 1 .. 31;
  type Day_Of_Month_Access is access all Day_Of_Month;

  procedure Free is new Ada.Unchecked_Deallocation
      (Object => Day_Of_Month
       Name   => Day_Of_Month_Access);

  A  : aliased Day_Of_Month;
  Ptr: Day_Of_Month_Access := A'Access;

begin

   Free(Ptr);

end;
```

With a simple **access** you know at least that you won't try to deallocate a stack object.

13.8.2 Access vs. System.Address

An access can be something different from a mere memory address, it may be something more. For example, an "access to String" often needs some way of storing the string size as well. If you need a simple address and are not concerned about strong typing, use the System.Address type.

13.8.3 C compatible pointer

The correct way to create a C compatible access is to use **pragma** :

```
type Day_Of_Month is range 1 .. 31;
for  Day_Of_Month'Size use Interfaces.C.int'Size;

pragma  (Convention => C,
                  Entity     => Day_Of_Month);
```

```
type Day_Of_Month_Access is access Day_Of_Month;

pragma  (Convention => C,
                  Entity     => Day_Of_Month_Access);
```

pragma should be used on any type you want to use in C. The compiler will warn you if the type cannot be made C compatible.

You may also consider the following - shorter - alternative when declaring Day_Of_Month:

```
type Day_Of_Month is new Interfaces.C.int range 1 .. 31;
```

Before you use access types in C, you should consider using the normal "in", "out" and "in out" modifiers. **pragma** and **pragma** know how parameters are usually passed in C and will use a pointer to pass a parameter automatically where C would have used them as well. Of course the RM contains precise rules on when to use a pointer for "in", "out", and "in out" - see "B.3 Interfacing with C ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-3.html} ".

13.8.4 Where is void*?

While actually a problem for "interfacing with C", here are some possible solutions:

```
procedure Test is

  subtype Pvoid is System.Address;

  -- the declaration in C looks like this:
  -- int C_fun(int *)
  function C_fun (pv: Pvoid) return Integer;
  pragma  (Convention    => C,
                Entity        => C_fun,     -- any Ada name
                External_Name => "C_fun");  -- the C name

  Pointer: Pvoid;

  Input_Parameter: aliased Integer := 32;
  Return_Value   : Integer;

begin

  Pointer      := Input_Parameter'Address;
  Return_Value := C_fun (Pointer);

end Test;
```

Less portable but perhaps more usable (for 32 bit CPUs):

```
type void is mod 2 ** 32;
for void'Size use 32;
```

With GNAT you can get 32/64 bit portability by using:

```
type void is mod System.Memory_Size;
for void'Size use System.Word_Size;
```

Closer to the true nature of void - pointing to an element of zero size is a pointer to a null record. This also has the advantage of having a representation for `void` and `void*`:

```
type Void is null record;
pragma  (C, Void);

type Void_Ptr is access all Void;
pragma  (C, Void_Ptr);
```

13.9 Thin and Fat Access Types

Thin and Fat Access Types

The difference between an access type and an address will be detailed in the following. The term *pointer* is used because this is usual terminology.

There is a predefined unit `System.Address_to_Access_Conversion` converting back and forth between access values and addresses. Use these conversions with care, as is explained below.

13.9.1 Thin Pointers

Thin pointers grant access to constrained subtypes.

```
type Int     is range -100 .. +500;
type Acc_Int is access Int;

type Arr     is array (1 .. 80) of Character;
type Acc_Arr is access Arr;
```

Objects of subtypes like these have a static size, so a simple address suffices to access them. In the case of arrays, this is generally the address of the first element.

For pointers of this kind, use of `System.Address_to_Access_Conversion` is safe.

13.9.2 Fat Pointers

```
type Unc     is array (Integer range <>) of Character;
type Acc_Unc is access Unc;
```

Objects of subtype `Unc` need a constraint, i.e. a start and a stop index, thus pointers to them need also to include those. So a simple address like the one of the first component is not sufficient. Note that A'Address is the same as A(A'First)'Address for any array object.

For pointers of this kind, `System.Address_to_Access_Conversion` will probably not work satisfactorily.

13.9.3 Example

```
CO: aliased Unc (-1 .. +1) := (-1 .. +1 => ' ');
UO: aliased Unc            := (-1 .. +1 => ' ');
```

Here, CO is a *nominally constrained* object, a pointer to it need not store the constraint, i.e. a thin pointer suffices. In contrast, UO is an object of a *nominally unconstrained* subtype, its *actual subtype* is constrained by the initial value.

```
A: Acc_Unc            := CO'Access;  -- illegal
B: Acc_Unc            := UO'Access;  -- OK
C: Acc_Unc (CO'Range) := CO'Access;  -- also illegal
```

The relevant paragraphs in the RM are difficult to understand. In short words:

An access type's target type is called the *designated subtype*, in our example `Unc`. RM 3.10.2(27.1/2) requires that `Unc_Acc`'s designated subtype statically match the *nominal subtype* of the object.

Now the nominal subtype of `CO` is the constrained anonymous subtype `Unc (-1 .. +1)`, the nominal subtype of `UO` is the unconstrained subtype `Unc`. In the illegal cases, the designated and nominal subtypes do not statically match.

13.10 See also

See also

13.10.1 Wikibook

- Ada Programming[7]
- Ada Programming/Types[8]

7 http://en.wikibooks.org/wiki/Ada%20Programming
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes

13.10.2 Ada Reference Manual

Ada 95

- 4.8 Allocators ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-4-8.html}
- 13.11 Storage Management ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-13-11.html}
- 13.11.2 Unchecked Storage Deallocation ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-13-11-2.html}
- 3.7 Discriminants ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-7.html}
- 3.10 Access Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-10.html}
- 6.1 Subprogram Declarations ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-6-1.html}
- B.3 Interfacing with C ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-B-3.html}

Ada 2005

- 4.8 Allocators ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-8.html}
- 13.11 Storage Management ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-13-11.html}
- 13.11.2 Unchecked Storage Deallocation ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-13-11-2.html}
- 3.7 Discriminants ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-7.html}
- 3.10 Access Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-10.html}
- 6.1 Subprogram Declarations ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-6-1.html}
- B.3 Interfacing with C ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-3.html}

13.10.3 Ada Quality and Style Guide

- 5.4.5 Dynamic Data ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-4-5.html}
- 5.9.2 Unchecked Deallocation ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-9-2.html}

14 Limited types

14.1 Limited Types

Limited Types

When a type is declared **limited** this means that objects of the type cannot be assigned values of the same type. An Object `b` of limited type `LT` cannot be copied into an object `a` of same type `LT`.

Additionally, there is no predefined equality operation for objects of a limited type.

The desired effects of declaring a type limited include prevention of shallow copying. Also, the (unique) identity of an object is retained: once declared, a name of a variable of type `LT` will continue to refer to the same object.

The following example will use a rather simplifying type `Boat`.

```
type Boat is limited private;

function Choose
  ( Load  :   Sailors_Units;
   Speed :   Sailors_Units)
   return  Boat;

procedure Set_Sail ( The_Boat :   in out Boat) ;
```

When we declare a variable to be of type `Boat`, its name will denote one boat from then on. Boats will not be copied into one another.

The full view of a boat might be implemented as a record such as

```
type Boat is limited record
   Max_Sail_Area :   Sailors_Units;
   Max_Freight   :   Sailors_Units;
   Sail_Area     :   Sailors_Units;
   Freight       :   Sailors_Units;
end record;
```

The `Choose` function returns a `Boat` object depending on the parameters `Load` and `Speed`. If we now declare a variable of type Boat we will be better off Choosing an initial Boat (or else we might be dropping into uninitialized waters!). But when we do so, the initialization looks suspiciously like assignment which is not available with limited types:

```
procedure Travel ( People :   Positive;  Average_Speed :   Sailors_Units)  is

   Henrietta :   Boat :=    --assignment?
      Choose
        ( Load  =>  People *  Average_Weight *  1.  5,
          Speed =>  Average_Speed *  1.  5) ;

begin
   Set_Sail ( Henrietta) ;
end Travel;
```

Fortunately, current Ada distinguishes initialization from copying. Objects of a limited type may be initialized by an initialization expression on the right of the delimiter := .

(Just to prevent confusion: The Ada Reference Manual discriminates between *assignment* and *assignment statement*, where assignment is part of the assignment statement. An initialisation is of course an assignment which, for limited types, is done *in place*. An assignment statement involves copying, which is forbidden for limited types.)

Related to this feature are aggregates of limited types[1] and "constructor functions" for limited types. Internally, the implementation of the `Choose` function will return a limited record. However, since the return type `Boat` is limited, there must be no copying anywhere. Will this work? A first attempt might be to declare a `result` variable local to `Choose`, manipulate `result`, and return it. The `result` object needs to be "transported" into the calling environment. But `result` is a variable local to `Choose`. When `Choose` returns, `result` will no longer be in scope. Therefore it looks like `result` must be copied but this is not permitted for limited types. There are two solutions provided by the language: extended return statements (see 6.5 Return Statements ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-6-5.html`}) and aggregates of limited types. The following body of `Choose` returns an aggregate of limited type `Boat`, after finding the initial values for its components.

```
function Choose
  ( Load  :   Sailors_Units;
    Speed :   Sailors_Units)
    return  Boat
is
   Capacity :   constant Sailors_Units :=  Capacity_Needed ( Load) ;
begin
   return Boat'
     ( Max_Freight   =>  Capacity,
       Max_Sail_Area =>  Sail_Needed ( Capacity) ,
       Freight       =>  Load,
       Sail_Area     =>  0.  0) ;
end Choose;
```

The object that is returned is at the same time the object that is to have the returned value. The function therefore initializes `Henrietta` *in place*.

In parallel to the predefined type Ada `.Controlled`, Ada provides the type `Limited_Controlled` in the same package. It is a limited version of the former.

1 `http://www.adacore.com/2007/05/14/gem-1/`

14.2 Initialising Limited Types

Initialising Limited Types

A few methods to initialise such types are presented.

```
package Limited_Private_Samples is

  type Uninitialised  is limited private;
  type Preinitialised is limited private;

  type Dynamic_Initialisation is limited private;
  function Constructor (X: Integer)  --any kind of parameters
    return Dynamic_Initialisation;

  type Needs_Constructor (<>) is limited private;
  function Constructor (X: Integer)  --any kind of parameters
    return Needs_Constructor;

private

  type Uninitialised is record
    I: Integer;
  end record;

  type Preinitialised is record
    I: Integer := 0;  --can also be a function call
  end record;

  type Void is null record;
  function Constructor (Object: access Dynamic_Initialisation) return Void;

  type Dynamic_Initialisation is limited record
    Hook: Void := Constructor (Dynamic_Initialisation'Access);
    Bla : Integer;  --any needed components
  end record;

  type Needs_Constructor is record
    I: Integer;
  end record;

end Limited_Private_Samples;
```

```
package body Limited_Private_Samples is

  function Constructor (Object: access Dynamic_Initialisation) return Void is
  begin
    Object.Bla := 5;  --may be any value only known at run time
    return (null record);
  end Constructor;

  function Constructor (X: Integer) return Dynamic_Initialisation is
  begin
    return (Hook => (null record),
            Bla  => 42);
  end Constructor;

  function Constructor (X: Integer) return Needs_Constructor is
  begin
    return (I => 42);
  end Constructor;

end Limited_Private_Samples;
```

```
with Limited_Private_Samples;
use  Limited_Private_Samples;

procedure Try is

  U:  Uninitialised;    --very bad
  P:  Preinitialised;   --has initial value (good)

  D1:   Dynamic_Initialisation;   --has initial value (good)
  D2:   Dynamic_Initialisation :=  Constructor ( 0) ;   --Ada 2005 initialisation
  D3:   Dynamic_Initialisation renames Constructor ( 0) ;   --already Ada 95

  --I: Needs_Constructor; -- Illegal without initialisation
  N:  Needs_Constructor :=  Constructor ( 0) ;   --Ada 2005 initialisation

begin

  null;

end Try;
```

Note that D3 is a constant, whereas all others are variables.

Also note that the initial value that is defined for the component of Preinitialised is evaluated at the time of object creation, i.e. if an expression is used instead of the literal, the value can be run-time dependent.

```
X, Y: Preinitialised;
```

In this declaration of two objects, the initial expression will be evaluated twice and can deliver different values, because it is equivalent to the sequence[2]:

```
X: Preinitialised;
Y: Preinitialised;
```

So X is initialised before Y.

14.3 See also

See also

14.3.1 Ada 95 Reference Manual

- 7.5 Limited Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-7-5.html}

2 ISO/IEC 8652:2007. 3.3.1 Object Declarations (7). Ada 2005 Reference Manual. Any declaration [...] with more than one `defining_identifier` is equivalent to a series of declarations each containing one `defining_identifier` from the list, [...] in the same order as the list. ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-3-1.html}

14.3.2 Ada 2005 Reference Manual

- 7.5 Limited Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-7-5.html}

14.3.3 Ada Quality and Style Guide

- 5.3.3 Private Types ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-3-3.html}
- 8.3.3 Formal Private and Limited Private Types ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_8/8-3-3.html}

14.4 References

References

15 Strings

Ada supports three different types of strings. Each string type is designed to solve a different problem.

In addition, every string type is implemented for each available Characters type (Character, Wide_Character, Wide_Wide_Character) giving a complement of nine combinations.

15.1 Fixed-length string handling

Fixed-length string handling

Fixed-Length Strings are arrays[1] of Character, and consequently of a fixed length. Since a fixed length string is an indefinite subtype[2] the length does not need to be known at compile time — the length may well be calculated at run time. In the following example the length is calculated from command-line argument 1:

```
X : String := Ada.Command_Line.Argument (1);
```

However once the length has been calculated and the strings have been created the length stays constant. Try the following program which shows a typical mistake:

File: show_commandline_1.adb

```
with Ada ;
with Ada ;

procedure Show_Commandline_1 is

   package T_IO renames Ada ;
   package CL   renames Ada ;

   X : String := CL.Argument (1);

begin
   T_IO.Put ("Argument 1 = ");
   T_IO.Put_Line (X);

   X := CL.Argument (2);

   T_IO.Put ("Argument 2 = ");
```

1 Chapter 11 on page 83

2 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23indefinite%20subtype

```
   T_IO.Put_Line (X);
end Show_Commandline_1;
```

The program will only work when the 1st and 2nd parameter have the same length. This is even true when the 2nd parameter is shorter. There is neither an automatic padding of shorter strings nor an automatic truncation of longer strings.

Having said that, the package Ada contains a set of procedures and functions for Fixed-Length String Handling which allows padding of shorter strings and truncation of longer strings.

Try the following example to see how it works:

File: show_commandline_2.adb

```
with Ada ;
with Ada ;
with Ada ;

procedure Show_Commandline_2 is

   package T_IO renames Ada ;
   package CL   renames Ada ;
   package S    renames Ada ;
   package SF   renames Ada ;

   X : String := CL.Argument (1);

begin
   T_IO.Put ("Argument 1 = ");
   T_IO.Put_Line (X);

  SF.Move (
     Source  => CL.Argument (2),
     Target  => X,
     Drop    => S.Right,
     Justify => S.Left,
     Pad     => S.Space);

   T_IO.Put ("Argument 2 = ");
   T_IO.Put_Line (X);
end Show_Commandline_2;
```

15.2 Bounded-length string handling

Bounded-length string handling

Bounded-Length Strings can be used when the maximum length of a string is known and/or restricted. This is often the case in database applications where only a limited number of characters can be stored.

Like Fixed-Length Strings the maximum length does not need to be known at compile time — it can also be calculated at runtime — as the example below shows:

File: show_commandline_3.adb

```
with Ada ;
with Ada ;
with Ada ;

procedure Show_Commandline_3 is

   package T_IO renames Ada.Text_IO;
   package CL   renames Ada.Command_Line;

   function Max_Length (
      Value_1 : Integer;
      Value_2 : Integer)
   return
      Integer
   is
      Retval : Integer;
   begin
      if Value_1 > Value_2 then
         Retval := Value_1;
      else
         Retval := Value_2;
      end if;
      return Retval;
   end Max_Length;

   pragma Inline (Max_Length);

   package SB
   is new Ada.Strings.Bounded.Generic_Bounded_Length (
       Max => Max_Length (
                  Value_1 => CL.Argument (1)'Length,
                  Value_2 => CL.Argument (2)'Length));

   X :  SB.Bounded_String
     := SB.To_Bounded_String (CL.Argument (1));

begin
   T_IO.Put ("Argument 1 = ");
   T_IO.Put_Line (SB.To_String (X));

   X := SB.To_Bounded_String (CL.Argument (2));

   T_IO.Put ("Argument 2 = ");
   T_IO.Put_Line (SB.To_String (X));
end Show_Commandline_3;
```

You should know that Bounded-Length Strings have some distinct disadvantages. Most noticeable is that each Bounded-Length String is a different type which makes converting them rather cumbersome. Also a Bounded-Length String type always allocates memory for the maximum permitted string length for the type. The memory allocation for a Bounded-Length String is equal to the maximum number of string "characters" plus an implementation dependent number containing the string length (each character can require allocation of more than one byte per character, depending on the underlying character type of the string,

and the length number is 4 bytes long for the Windows GNAT Ada compiler v3.15p, for example).

15.3 Unbounded-length string handling

Unbounded-length string handling

Last but not least there is the Unbounded-Length String. In fact: If you are not doing embedded or database programming this will be the string type you are going to use most often as it gives you the maximum amount of flexibility.

As the name suggest the Unbounded-Length String can hold strings of almost any length — limited only to the value of Integer'Last or your available heap memory. This is because Unbounded_String type is implemented using dynamic memory allocation behind the scenes, providing lower efficiency but maximum flexibility.

File: show_commandline_4.adb

```
with Ada ;
with Ada ;
with Ada ;

procedure Show_Commandline_4 is

   package T_IO renames Ada.Text_IO;
   package CL   renames Ada.Command_Line;
   package SU   renames Ada.Strings.Unbounded;

   X :  SU.Unbounded_String
     := SU.To_Unbounded_String (CL.Argument (1));

begin
   T_IO.Put ("Argument 1 = ");
   T_IO.Put_Line (SU.To_String (X));

   X := SU.To_Unbounded_String (CL.Argument (2));

   T_IO.Put ("Argument 2 = ");
   T_IO.Put_Line (SU.To_String (X));
end Show_Commandline_4;
```

As you can see the Unbounded-Length String example is also the shortest (discarding the first example, which is buggy) — this makes using Unbounded-Length Strings very appealing.

15.4 See also

See also

15.4.1 Wikibook

- Ada Programming[3]

15.4.2 Ada 95 Reference Manual

- 2.6 String Literals ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-2-6.html}
- 3.6.3 String Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-6-3.html}
- A.4.3 Fixed-Length String Handling ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-A-4-3.html}
- A.4.4 Bounded-Length String Handling ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-A-4-4.html}
- A.4.5 Unbounded-Length String Handling ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-A-4-5.html}

15.4.3 Ada 2005 Reference Manual

- 2.6 String Literals ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-6.html}
- 3.6.3 String Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-6-3.html}
- A.4.3 Fixed-Length String Handling ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-4-3.html}
- A.4.4 Bounded-Length String Handling ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-4-4.html}
- A.4.5 Unbounded-Length String Handling ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-4-5.html}

3 http://en.wikibooks.org/wiki/Ada%20Programming

16 Subprograms

In Ada the subprograms are classified into two categories: procedures[1] and functions[2]. A procedures call is a statement and does not return any value, whereas a function returns a value and must therefore be a part of an expression.

Subprogram parameters may have three modes.

in

The actual parameter value goes into the call and is not changed there. The formal parameter is a constant and allows only reading. This is the default when no mode is given. The actual parameter is an expression.

in out

The actual parameter goes into the call and may be redefined. The formal parameter is a variable and can be read and written.

out

The actual parameter's value before the call is irrelevant, it will get a value in the call. The formal parameter can be read and written. (In Ada 83 **out** parameters were write-only.)

A parameter of any mode may also be explicitly **aliased**.

access

The formal parameter is an access (a pointer) to some variable. (This is not a parameter mode from the reference manual point of view.)

Note that parameter modes do not specify the parameter passing method. Their purpose is to document the data flow.

The parameter passing method depends on the type of the parameter. A rule of thumb is that parameters fitting into a register are passed by copy, others are passed by reference. For certain types, there are special rules, for others the parameter passing mode is left to the compiler (which you can assume to do what is most sensible). Tagged types are always passed by reference.

Explicitly **aliased** parameters and **access** parameters specify pass by reference.

Unlike in the C class of programming languages, Ada subprogram calls cannot have empty parameters parentheses `( )` when there are no parameters.

1 Chapter 16.1 on page 126
2 Chapter 16.2 on page 127

16.1 Procedures

Procedures

A procedure call in Ada constitutes a statement by itself.

For example:

```
procedure A_Test ( A, B: in Integer; C: out Integer)  is
begin
   C := A + B;
end A_Test;
```

When the procedure is called with the statement

```
A_Test (5 + P, 48, Q);
```

the expressions 5 + P and 48 are evaluated (expressions are only allowed for in parameters), and then assigned to the formal parameters A and B, which behave like constants. Then, the value A + B is assigned to formal variable C, whose value will be assigned to the actual parameter Q when the procedure finishes.

C, being an **out** parameter, is an uninitialized variable before the first assignment. (Therefore in Ada 83, there existed the restriction that **out** parameters are write-only. If you wanted to read the value written, you had to declare a local variable, do all calculations with it, and finally assign it to C before return. This was awkward and error prone so the restriction was removed in Ada 95.)

Within a procedure, the return statement can be used without arguments to exit the procedure and return the control to the caller.

For example, to solve an equation of the kind $ax^2 + bx + c = 0$:

```
with Ada ;
use  Ada ;

procedure Quadratic_Equation
   ( A, B, C :     Float;   --By default it is "in".
    R1, R2  : out Float;
    Valid   : out Boolean)
is
   Z : Float;
begin
   Z := B**2 - 4.0 * A * C;
   if Z < 0.0 or A = 0.0 then
      Valid := False;  --Being out parameter, it should be modified at least once.
      R1    := 0.0;
      R2    := 0.0;
   else
      Valid := True;
      R1    := (-B + Sqrt (Z)) / (2.0 * A);
      R2    := (-B - Sqrt (Z)) / (2.0 * A);
   end if;
end Quadratic_Equation;
```

The function SQRT calculates the square root of non-negative values. If the roots are real, they are given back in R1 and R2, but if they are complex or the equation degenerates (A = 0), the execution of the procedure finishes after assigning to the Valid variable the False value, so that it is controlled after the call to the procedure. Notice that the **out** parameters should be modified at least once, and that if a mode is not specified, it is implied **in**.

16.2 Functions

Functions

A function is a subprogram that can be invoked as part of an expression. Until Ada 2005, functions can only take **in** (the default) or **access** parameters; the latter can be used as a work-around for the restriction that functions may not have **out** parameters. Ada 2012 has removed this restriction.

Here is an example of a function body:

```
function Minimum (A, B: Integer) return Integer is
begin
   if A <= B then
      return A;
   else
      return B;
   end if;
end Minimum;
```

Or in Ada2012:

```
function Minimum (A, B: Integer) return Integer is
begin
   return (if A <= B then A else B);
end Minimum;
```

or even shorter as an *expression function*

```
function Minimum (A, B: Integer) return Integer is (if A <= B then A else B);
```

The formal parameters with mode **in** behave as local constants whose values are provided by the corresponding actual parameters. The statement **return** is used to indicate the value returned by the function call and to give back the control to the expression that called the function. The expression of the **return** statement may be of arbitrary complexity and must be of the same type declared in the specification. If an incompatible type is used, the compiler gives an error. If the restrictions of a subtype are not fulfilled, e.g. a range, it raises a Constraint_Error exception.

The body of the function can contain several **return** statements and the execution of any of them will finish the function, returning control to the caller. If the flow of control within the function branches in several ways, it is necessary to make sure that each one of them is finished with a **return** statement. If at run time the end of a function is reached without

encountering a **return** statement, the exception Program_Error is raised. Therefore, the body of a function must have at least one such **return** statement.

Every call to a function produces a new copy of any object declared within it. When the function finalizes, its objects disappear. Therefore, it is possible to call the function recursively. For example, consider this implementation of the factorial function:

```
function Factorial (N : Positive) return Positive is
begin
   if N = 1 then
      return 1;
   else
      return (N * Factorial (N - 1));
   end if;
end Factorial;
```

When evaluating the expression `Factorial (4);` the function will be called with parameter 4 and within the function it will try to evaluate the expression `Factorial (3)`, calling itself as a function, but in this case parameter N would be 3 (each call copies the parameters) and so on until N = 1 is evaluated which will finalize the recursion and then the expression will begin to be completed in the reverse order.

A formal parameter of a function can be of any type, including vectors or records. Nevertheless, it cannot be an anonymous type, that is, its type must be declared before, for example:

```
type Float_Vector is array (Positive range <>) of Float;

function Add_Components (V: Float_Vector) return Float is
   Result : Float := 0.0;
begin
   for I in V'Range loop
      Result := Result + V(I);
   end loop;
   return Result;
end Add_Components;
```

In this example, the function can be used on a vector of arbitrary dimension. Therefore, there are no static bounds in the parameters passed to the functions. For example, it is possible to be used in the following way:

```
V4  : Float_Vector (1 .. 4) := (1.2, 3.4, 5.6, 7.8);
Sum : Float;

Sum := Add_Components (V4);
```

In the same way, a function can also return a type whose bounds are not known a priori. For example:

```
function Invert_Components (V : Float_Vector) return Float_Vector is
   Result : Float_Vector(V'Range);   --Fix the bounds of the vector to be returned.
begin
   for I in V'Range loop
```

```
      Result(I) := V (V'First + V'Last - I);
   end loop;
   return Result;
end Invert_Components;
```

The variable Result has the same bounds as V, so the returned vector will always have the same dimension as the one passed as parameter.

A value returned by a function can be used without assigning it to a variable, it can be referenced as an expression. For example, `Invert_Components (V4) (1)`, where the first element of the vector returned by the function would be obtained (in this case, the last element of V4, i.e. 7.8).

16.3 Named parameters

Named parameters

In subprogram calls, named parameter notation (i.e. the name of the formal parameter followed of the symbol => and then the actual parameter) allows the rearrangement of the parameters in the call. For example:

```
Quadratic_Equation (Valid => OK, A => 1.0, B => 2.0, C => 3.0, R1 =>
P, R2 => Q);
F := Factorial (N => (3 + I));
```

This is especially useful to make clear which parameter is which.

```
Phi := Arctan (A, B);
Phi := Arctan (Y => A, X => B);
```

The first call (from Ada.Numerics.Elementary_Functions) is not very clear. One might easily confuse the parameters. The second call makes the meaning clear without any ambiguity.

Another use is for calls with numeric literals:

```
Ada.Float_Text_IO.Put_Line (X, 3, 2, 0);  -- ?
Ada.Float_Text_IO.Put_Line (X, Fore => 3, Aft => 2, Exp => 0);  --
OK
```

16.4 Default parameters

Default parameters

On the other hand, formal parameters may have default values. They can, therefore, be omitted in the subprogram call. For example:

```
procedure By_Default_Example (A, B: in Integer := 0);
```

can be called in these ways:

```
By_Default_Example (5, 7);      --A = 5, B = 7
By_Default_Example (5);         --A = 5, B = 0
By_Default_Example;             --A = 0, B = 0
By_Default_Example (B => 3);    --A = 0, B = 3
By_Default_Example (1, B => 2); --A = 1, B = 2
```

In the first statement, a "regular call" is used (with positional association); the second also uses positional association but omits the second parameter to use the default; in the third statement, all parameters are by default; the fourth statement uses named association to omit the first parameter; finally, the fifth statement uses mixed association, here the positional parameters have to precede the named ones.

Note that the default expression is evaluated once for each formal parameter that has no actual parameter. Thus, if in the above example a function would be used as defaults for A and B, the function would be evaluated once in case 2 and 4; twice in case 3, so A and B could have different values; in the others cases, it would not be evaluated.

16.5 Renaming

Renaming

Subprograms may be renamed. The parameter and result profile for a renaming-as-declaration must be mode conformant.

```
procedure Solve
  (A, B, C: in  Float;
   R1, R2 : out Float;
   Valid  : out Boolean) renames Quadratic_Equation;
```

This may be especially comfortable for tagged types.

```
package Some_Package is
  type Message_Type is tagged null record;
  procedure Print (Message: in Message_Type);
end Some_Package;
```

```
with Some_Package;
procedure Main is
  Message: Some_Package.Message_Type;
  procedure Print renames Message.Print;  --this has convention intrinsic, see RM 6.3.1(10.1/2)
  Method_Ref:  access procedure := Print'Access;  --thus taking 'Access should be illegal; GNAT
GPL 2012 allows this
begin  --All these calls are equivalent:
  Some_Package.Print (Message);  --traditional call without use clause
  Message.Print;                 --Ada 2005 method.object call - note:  no use clause necessary
  Print;                         --Message.Print is a parameterless procedure and can be renamed
```

```
as such
   Method_Ref.all;                --GNAT GPL 2012 allows illegal call via an access to the renamed
procedure Print
                                  --This has been corrected in the current version (as of Nov 22,
2012)
 end Main;
```

But note that `Message.Print'Access;` is illegal, you have to use a renaming declaration as above.

Since only mode conformance is required (and not full conformance as between specification and body), parameter names and default values may be changed with renamings:

```
procedureP (X: inInteger :=  0);
procedureR (A: inInteger := -1) renamesP;
```

16.6 See also

See also

16.6.1 Wikibook

- Ada Programming[3]
- Ada Programming/Operators[4]

16.6.2 Ada 95 Reference Manual

- Section 6: Subprograms ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-6.html}
- 4.4 Expressions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-4.html}

16.6.3 Ada 2005 Reference Manual

- Section 6: Subprograms ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-6.html}
- 4.4 Expressions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-4.html}

3 http://en.wikibooks.org/wiki/Ada%20Programming
4 Chapter 37 on page 301

16.6.4 Ada Quality and Style Guide

- 4.1.3 Subprograms ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_4/4-1-3.html}

es:Programación en Ada/Subprogramas[5]

5 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FSubprogramas

17 Packages

Ada encourages the division of code into separate modules called *packages*. Each package can contain any combination of items.

Some of the benefits of using packages are:

- package contents are placed in a separate namespace, preventing naming collisions,
- implementation details of the package can be hidden from the programmer (information hiding),
- object orientation requires defining a type and its primitive subprograms within a package, and
- packages can be separately compiled.

Some of the more common package usages are:

- a group of related subprograms along with their shared data, with the data not visible outside the package,
- one or more data types along with subprograms for manipulating those data types, and
- a generic package that can be instantiated under varying conditions.

The following is a quote from the current Ada Reference Manual 7. Packages. RM 7(1) `^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-7-.html}`

> Packages are program units that allow the specification of groups of logically related entities. Typically, a package contains the declaration of a type (often a private type or private extension) along with the declaration of primitive subprograms of the type, which can be called from outside the package, while their inner workings remain hidden from outside users.

17.1 Separate compilation

Separate compilation

It is very common for package declarations and package bodies to be coded into separate files and separately compiled. Doing so places the package at the *library level* where it will be accessible to all other code via the **with** statement—if a more restricted scope is desired, simply declare the package (and package body, if needed) within the appropriate scope. The package body can itself be divided into multiple files by specifying that one or more subprogram implementations are **separate**.

One of the biggest advantages of Ada over most other programming languages is its well defined system of modularization and separate compilation. Even though Ada allows separate compilation, it maintains the strong type checking among the various compilations by enforcing rules of compilation order and compatibility checking. Ada uses separate compilation (like Modula-2[1], Java[2] and C#[3]), and not independent compilation (as C[4]/C++[5] does), in which the various parts are compiled with no knowledge of the other compilation units with which they will be combined.

A note to C/C++ users: Yes, you can use the preprocessor to emulate separate compilation — but it is only an emulation and the smallest mistake leads to very hard to find bugs. It is telling that all C/C++ successor languages including D[6] have turned away from the independent compilation and the use of the preprocessor.

So it's good to know that Ada has had separate compilation ever since Ada-83 and is probably the most sophisticated implementation around.

17.2 Parts of a package

Parts of a package

A package generally consists of two parts, the specification and the body. A package specification can be further divided in two logical parts, the visible part and the private part. Only the visible part of the specification is mandatory. The private part of the specification is optional, and a package specification might not have a package body—the package body only exists to complete any *incomplete* items in the specification. Subprogram declarations are the most common *incomplete* items. There must not be a package body if there is no incomplete declarations and there has to be a package body if there is incomplete declarations in the specification.

To understand the value of the three-way division, consider the case of a package that has already been released and is in use. A change to the visible part of the specification will require that the programmers of all using software verify that the change does not affect the using code. A change to the private part of the declaration will require that all using code be recompiled but no review is normally needed. Some changes to the private part can change the meaning of the client code however. An example is changing a private record type into a private access type. This change can be done with changes in the private part, but change the semantic meaning of assignment in the clients code. A change to the package body will only require that the file containing the package body be recompiled, because *nothing* outside of the package body can ever access anything within the package body (beyond the declarations in the specification part).

A common usage of the three parts is to declare the existence of a record type and some subprograms that operate on that type in the visible part, define the actual structure of the

1 http://en.wikipedia.org/wiki/Modula-2
2 http://en.wikipedia.org/wiki/Java_programming_language
3 http://en.wikipedia.org/wiki/C%20Sharp%20programming%20language
4 http://en.wikipedia.org/wiki/C%20programming%20language
5 http://en.wikipedia.org/wiki/C%2B%2B
6 http://en.wikipedia.org/wiki/D_programming_language

record type in the private part, and provide the code to implement the subprograms in the package body.

17.2.1 The package specification — the visible part

The visible part of a package specification describes all the subprogram specifications, variables, types, constants etc. that are visible to anyone who wishes to use the package.

```
package Public_Only_Package is

  type Range_10 is range 1 .. 10;

end Public_Only_Package;
```

17.2.2 The private part

The private part of a package serves two purposes:

- To complete the deferred definition of private types and constants.
- To export entities only visible to the children of the package

```
package Package_With_Private is

   type Private_Type is private;

private

   type Private_Type is array (1 .. 10) of Integer;

end Package_With_Private;
```

17.2.3 The package body

The package body defines the implementation of the package. All the subprograms defined in the specification have to be implemented in the body. New subprograms, types and objects can be defined in the body that are not visible to the users of the package.

```
package Package_With_Body is

   type Basic_Record is private;

   procedure Set_A (This : in out Basic_Record;
                    An_A : in     Integer);

   function Get_A (This : Basic_Record) return Integer;

private

   type Basic_Record is
      record
         A : Integer;
```

```
      end record ;

   pragma   (Get_A);
   pragma  (Get_A);
   pragma  (Set_A);

end Package_With_Body;
```

```
package body Package_With_Body is

   procedure Set_A (This : in out Basic_Record;
                    An_A : in     Integer)
   is
   begin
      This.A := An_A;
   end Set_A;

   function Get_A (This : Basic_Record) return Integer is
   begin
      return This.A;
   end Get_A;

end Package_With_Body;
```

`pragma`

Only available when using GNAT[7].

17.2.4 Two Flavors of Package

The packages above each define a type together with operations of the type. When the type's composition is placed in the private part of a package, the package then exports what is known to be an Abstract Data Type[8] or ADT for short. Objects of the type are then constructed by calling one of the subprograms associated with the respective type.

A different kind of package is the Abstract State Machine. A package will be modeling a single item of the problem domain, such as the motor of a car. If a program controls one car, there is typically just one motor, or *the* motor. The public part of the package specification only declares the operations of the module (of the motor, say), but no type. All data of the module are hidden in the body of the package where they act as state variables to be queried, or manipulated by the subprograms of the package. The initialization part sets the state variables to their initial values.

```
package Package_With_Body is

   procedure Set_A ( An_A :   in Integer) ;

   function Get_A return Integer;

private
```

7 http://en.wikipedia.org/wiki/GNAT
8 http://en.wikipedia.org/wiki/Abstract_data_type

```
   pragma Pure_Function ( Get_A) ;

end Package_With_Body;
```

```
package body Package_With_Body is

   The_A:  Integer;

   procedure Set_A ( An_A :   in Integer)  is
   begin
      The_A :=  An_A;
   end Set_A;

   function Get_A return Integer is
   begin
      return The_A;
   end Get_A;


begin

   The_A :=  0;

end Package_With_Body;
```

(A note on construction: The package initialization part after **begin** corresponds to a construction subprogram of an ADT package. However, as a state machine *is* an "object" already, "construction" is happening during package initialization. (Here it sets the state variable `The_A` to its initial value.) An ASM package can be viewed as a singleton[9].)

17.3 Using packages

Using packages

To utilize a package it's needed to name it in a **with** clause, whereas to have direct visibility of that package it's needed to name it in a **use** clause.

For C++ programmers, Ada's **with** clause is analogous to the C++ preprocessor's **#include** and Ada's **use** is similar to the **using namespace** statement in C++. In particular, **use** leads to the same namespace pollution problems as **using namespace** and thus should be used sparingly. Renaming can shorten long compound names to a manageable length, while the **use type** clause makes a type's operators visible. These features reduce the need for plain **use**.

17.3.1 Standard with

The standard with clause provides visibility for the public part of a unit to the following defined unit. The imported package can be used in any part of the defined unit, including the body when the clause is used in the specification.

9 http://en.wikipedia.org/wiki/Singleton_pattern

17.3.2 Private with

This language feature is only available in Ada 2005

```
private with Ada.Strings.Unbounded;

package Private_With is

   --The package Ada.String.Unbounded is not visible at this point

   type Basic_Record is private;

   procedure Set_A (This : in out Basic_Record;
                    An_A : in     String);

   function Get_A (This : Basic_Record) return String;

private
   --The visibility of package Ada.String.Unbounded starts here

   package Unbounded renames Ada.Strings.Unbounded;

   type Basic_Record is
      record
         A : Unbounded.Unbounded_String;
      end record;

   pragma   (Get_A);
   pragma  (Get_A);
   pragma  (Set_A);

end Private_With;
```

```
package body Private_With is

   --The private withed package is visible in the body too

   procedure Set_A (This : in out Basic_Record;
                    An_A : in     String)
   is
   begin
      This.A := Unbounded.To_Unbounded_String (An_A);
   end Set_A;

   function Get_A (This : Basic_Record) return String is
   begin
      return Unbounded.To_String (This.A);
   end Get_A;

end Private_With;
```

17.3.3 Limited with

This language feature is only available in Ada 2005

```
limited with Departments;

package Employees is
```

```
   type Employee is tagged private;

   procedure Assign_Employee
     (E : in out Employee;
      D : access Departments.Department'Class);

   type Dept_Ptr is access all Departments.Department'Class;

   function Current_Department(E : in Employee) return Dept_Ptr;
   ...
end Employees;
```

```
limited with Employees;

package Departments is

   type Department is tagged private;

   procedure Choose_Manager
     (Dept    : in out Department;
      Manager : access Employees.Employee'Class);
   ...
end Departments;
```

17.3.4 Making operators visible

Suppose you have a package Universe that defines some numeric type T.

```
with Universe;
procedure P is
  V:  Universe.  T :=  10.  0;
begin
  V :=  V *  42.  0;   --illegal
end P;
```

This program fragment is illegal since the operators implicitly defined in Universe are not directly visible.

You have four choices to make the program legal.

Use a use_package_clause. This makes **all declarations** in Universe directly visible (provided they are not hidden because of other homographs).

```
with Universe;
use  Universe;
procedure P is
  V:  Universe.  T :=  10.  0;
begin
  V :=  V *  42.  0;
end P;
```

Use renaming. This is error prone since if you rename many operators, cut and paste errors are probable.

```
with Universe;
procedure P is
  function "*" ( Left,  Right:   Universe.  T)  return Universe.  T renames Universe.  "*";
  function "/" ( Left,  Right:   Universe.  T)  return Universe.  T renames Universe.  "*";
 --oops
  V:  Universe.  T :=  10.  0;
begin
  V :=  V *  42.  0;
end P;
```

Use qualification. This is extremely ugly and unreadable.

```
with Universe;
procedure P is
  V:  Universe.  T :=  10.  0;
begin
  V :=  Universe.  "*" ( V,  42.  0) ;
end P;
```

Use the use_type_clause. This makes only the **operators** in Universe directly visible.

```
with Universe;
procedure P is
  V:  Universe.  T :=  10.  0;
  use type Universe.  T;
begin
  V :=  V *  42.  0;
end P;
```

There is a special beauty in the use_type_clause. Suppose you have a set of packages like so:

```
with Universe;
package Pack is
  subtype T is Universe.  T;
end Pack;
```

```
with Pack;
procedure P is
  V:  Pack.  T :=  10.  0;
begin
  V :=  V *  42.  0;   --illegal
end P;
```

Now you've got into trouble. Since Universe is not made visible, you cannot use a use_package_clause for Universe to make the operator directly visible, nor can you use qualification for the same reason. Also a use_package_clause for Pack does not help, since the operator is not defined in Pack. The effect of the above construct means that the operator is not nameable, i.e. it cannot be renamed in a renaming statement.

Of course you can add Universe to the context clause, but this may be impossible due to some other reasons (e.g. coding standards); also adding the operators to Pack may be forbidden or not feasible. So what to do?

The solution is simple. Use the use_type_clause for Pack.T and all is well!

```
with Pack;
procedure P is
  V:  Pack.  T :=  10.  0;
  use type Pack.  T;
begin
  V :=  V *  42.  0;
end P;
```

17.4 Package organisation

Package organisation

17.4.1 Nested packages

A nested package is a package declared inside a package. Like a normal package, it has a public part and a private part. From outside, items declared in a nested package N will have visibility as usual; the programmer may refer to these items using a full dotted name like `P.N.X`. (But not `P.M.Y`.)

```
package P is
   D:  Integer;

   --a nested package:
   package N is
      X:  Integer;
   private
      Foo:   Integer;
   end N;

   E:  Integer;
private
   --another nested package:
   package M is
      Y:  Integer;
   private
      Bar:   Integer;
   end M;

end P;
```

Inside a package, declarations become visible as they are introduced, in textual order. That is, a nested package N that is declared *after* some other declaration D can refer to this declaration D. A declaration E following N can refer to items of N[10]. But neither can "look ahead" and refer to any declaration that goes after them. For example, spec `N` above cannot refer to `M` in any way.

In the following example, a type is derived in both of the two nested packages `Disks` and `Books`. Notice that the full declaration of parent type `Item` appears before the two nested

10 For example, `E: Integer := D + N.X;`

packages.

```
with Ada ;  use Ada ;

package Shelf is

   pragma Elaborate_Body;

   --things to put on the shelf

   type ID is range 1_000 ..   9_999;
   type Item ( Identifier :   ID)  is abstract tagged limited null record;
   type Item_Ref is access constant Item' class;

   function Next_ID return ID;
   --a fresh ID for an Item to Put on the shelf


   package Disks is

      type Music is (
         Jazz,
         Rock,
         Raga,
         Classic,
         Pop,
         Soul) ;

      type Disk ( Style :   Music;  Identifier :   ID)  is new Item ( Identifier)
         with record
            Artist :   Unbounded_String;
            Title  :   Unbounded_String;
         end record;

   end Disks;


   package Books is

      type Literature is (
         Play,
         Novel,
         Poem,
         Story,
         Text,
         Art) ;

      type Book ( Kind :   Literature;  Identifier :   ID)  is new Item
( Identifier)
         with record
            Authors :   Unbounded_String;
            Title   :   Unbounded_String;
            Year    :   Integer;
         end record;

   end Books;

   --shelf manipulation

   procedure Put ( it:   Item_Ref) ;
   function Get ( identifier :   ID)  return Item_Ref;
   function Search ( title :   String)  return ID;

private
```

```
   --keeping private things private

   package Boxes is
      type Treasure( Identifier:   ID)  is limited private;
   private
      type Treasure( Identifier:   ID)  is new Item( Identifier)  with null record;
   end Boxes;

end Shelf;
```

A package may also be nested inside a subprogram. In fact, packages can be declared in any declarative part, including those of a block.

17.4.2 Child packages

Ada allows one to extend the functionality of a unit (package) with so-called children (child packages). With certain exceptions, all the functionality of the parent is available to a child. This means that all public and private declarations of the parent package are visible to all child packages.

The above example, reworked as a hierarchy of packages, looks like this. Notice that the package Ada is not needed by the top level package `Shelf`, hence its with clause doesn't appear here. (We have added a match function for searching a shelf, though):

```
package Shelf is

   pragma Elaborate_Body;

   type ID is range 1_000 ..   9_999;
   type Item ( Identifier :   ID)  is abstract tagged limited null record;
   type Item_Ref is access constant Item' Class;

   function Next_ID return ID;
   --a fresh ID for an Item to Put on the shelf

   function match ( it :   Item;  Text :   String)  return Boolean is abstract;
   --see whether It has bibliographic information matching Text


   --shelf manipulation

   procedure Put ( it:   Item_Ref) ;
   function Get ( identifier :   ID)  return Item_Ref;
   function Search ( title :   String)  return ID;

end Shelf;
```

The name of a child package consists of the parent unit's name followed by the child package's identifier, separated by a period (dot) '. '.

```
with Ada ;  use Ada ;

package Shelf.  Books is

   type Literature is (
      Play,
```

```
      Novel,
      Poem,
      Story,
      Text,
      Art) ;

   type Book ( Kind :   Literature;  Identifier :   ID)  is new Item ( Identifier)
      with record
         Authors :   Unbounded_String;
         Title   :   Unbounded_String;
         Year    :   Integer;
      end record;

   function match( it:   Book;  text:   String)  return Boolean;

end Shelf.  Books;
```

`Book` has two components of type `Unbounded_String`, so Ada appears in a with clause of the child package. This is unlike the nested packages case which requires that all units needed by any one of the nested packages be listed in the context clause of the enclosing package (see 10.1.2 Context Clauses - With Clauses ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-10-1-2.html`}). Child packages thus give better control over package dependences. With clauses are more local.

The new child package `Shelf.Disks` looks similar. The `Boxes` package which was a nested package in the private part of the original `Shelf` package is moved to a private child package:

```
private package Shelf.  Boxes is
    type Treasure( Identifier:   ID)  is limited private;
private
    type Treasure( Identifier:   ID)  is new Item( Identifier)  with null record;
    function match( it:   Treasure;  text:   String)  return Boolean;
end Shelf.  Boxes;
```

The privacy of the package means that it can only be used by equally private client units. These clients include private siblings and also the bodies of siblings (as bodies are never public).

Child packages may be listed in context clauses just like normal packages. A **with** of a child also 'withs' the parent.

17.4.3 Subunits

A subunit is just a feature to move a body into a place of its own when otherwise the enclosing body will become too large. It can also be used for limiting the scope of context clauses.

The subunits allow to physically divide a package into different compilation units without breaking the logical unity of the package. Usually each separated subunit goes to a different file allowing separate compilation of each subunit and independent version control history for each one.

```
package body Pack is
  procedure Proc is separate;
end Pack;

with Some_Unit;
separate ( Pack)
procedure Proc is
begin
  ...
end Proc;
```

17.5 Notes

Notes

17.6 See also

See also

17.6.1 Wikibook

- Ada Programming[11]

17.6.2 Wikipedia

- Module[12]

17.6.3 Ada 95 Reference Manual

- Annex 7: Packages[13]

17.6.4 Ada 2005 Reference Manual

- Annex 7: Packages[14]

11 http://en.wikibooks.org/wiki/Ada%20Programming
12 http://en.wikipedia.org/wiki/Module%20%28programming%29
13 http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-7.html
14 http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-7.html

18 Input Output

18.1 Overview

Overview

The standard Ada libraries provide several Input/Output facilities, each one adapted to specific needs. Namely, the language defines the following dedicated packages:

- Text_IO
- Sequential_IO
- Direct_IO
- Stream_IO

The programmer must choose the adequate package depending on the application needs. For example, the following properties of the data handled by the application should be considered:

- **Data contents**: plain text, or binary data?
- **Accessing the data**: random access, or sequential access?
- **Medium**: data file, console, network/data-bus?
- **Data structure**: homogeneous file (sequence of the same data field), heterogeneous file (different data fields)?
- **Data format**: adherence to an existing data format, or the application can freely choose a new one?

For example, Stream_IO is very powerful and can handle complex data structures but can be heavier than other packages; Sequential_IO is lean and easy to use but cannot be used by applications requiring random data access; Text_IO can handle just textual data, but it is enough for handling the command-line console.

The following table gives some advices for choosing the more adequate one:

Simple heuristics for choosing an I/O package			
Data access	**Plain text**	**Binary data**	
		Homogeneous	Heterogeneous
Sequential	Text_IO	Sequential_IO	Stream_IO
Random	Text_IO	Direct_IO	Stream_IO

So the most important lesson to learn is choosing the right one. This chapter will describe more in detail these standard packages, explaining how to use them effectively. Besides these Ada-defined packages for general I/O operations each Ada compiler usually has other

implementation-defined Input-Output facilities, and there are also other external libraries for specialized I/O needs[1] like XML processing or interfacing with databases.

18.2 Text I/O

Text I/O

Text I/O[2] is probably the most used Input/Output package. All data inside the file are represented by human readable text. Text I/O provides support for line and page layout but the standard is free form text.

```
with Ada ;
use  Ada ;
with Ada ;
use  Ada ;

procedure Main is
  Str  :   String ( 1..  5) ;
  Last :   Natural;
begin
  Ada.  Text_IO. Get_Line ( Str,  Last) ;
  Ada.  Text_IO. Put_Line ( Str ( 1..  Last) ) ;
end;
```

It also contains several generic packages for converting numeric and enumeration types to character strings, or for handling Bounded and Unbounded strings, allowing the programmer to read and write different data types in the same file easily (there are ready-to-use instantiations of these generic packages for the Integer, Float, and Complex types). Finally, the same family of Ada.Text_IO packages (including the several children and instantiation packages) for the type Wide_Character and Wide_Wide_Character.

It is worth noting that the family of Text_IO packages provide some automatic text processing. For example, the Get_Line ignores white spaces at the beginning of a line (Get_Immediate does not present this behavior), or adding a newline character when closing the file. This is thus adequate for applications handling simple textual data, but users requiring direct management of text (e.g. raw access to the character encoding) must consider other packages like Sequential_IO.

18.3 Direct I/O

Direct I/O

Direct I/O is used for random access files which contain only elements of one specific type. With Direct_IO you can position the file pointer to any element of that type (random access), however you can't freely choose the element type, the element type needs to be a definite subtype[3].

1 http://en.wikibooks.org/wiki/Ada%20Programming%23Other%20Language%20Libraries
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO
3 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23definite_subtype

18.4 Sequential I/O

Sequential I/O

Direct I/O is used for random access files which contain only elements of one specific type. With Sequential_IO it is the other way round: you can choose between definite[4] and indefinite[5] element types but you have to read and write the elements one after the other.

18.5 Stream I/O

Stream I/O

Stream I/O is the most powerful input/output package which Ada provides. Stream I/O allows you to mix objects from different element types in one sequential file. In order to read/write from/to a stream each type provides a 'Read[6] and 'Write[7] attribute as well as an 'Input[8] and 'Output[9] attribute. These attributes are automatically generated for each type you declare.

The 'Read[10] and 'Write[11] attributes treat the elements as raw data. They are suitable for low level input/output as well as interfacing with other programming languages.

The 'Input[12] and 'Output[13] attribute add additional control informations to the file, like for example the 'First[14] and 'Last[15] attributes from an array.

In object orientated programming you can also use the 'Class[16]'Input[17] and 'Class[18]'Output[19] attributes - they will store and recover the actual object type as well.

Stream I/O is also the most flexible input/output package. All I/O attributes can be replaced with user defined functions or procedures using representation clauses and you can provide your own Stream I/O types using flexible object oriented techniques.

18.6 See also

4 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23definite_subtype
5 http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23Ada%20Programming%2FSubtypes%23indefinite_subtype
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Read
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Write
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Input
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Output
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Read
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Write
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Input
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Output
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27First
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Last
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Class
17 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Input
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Class
19 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Output

See also

18.6.1 Wikibook

- Ada Programming[20]
- Ada Programming/Libraries/Ada.Direct_IO[21]
- Ada Programming/Libraries/Ada.Sequential_IO[22]
- Ada Programming/Libraries/Ada.Streams[23]
  - Ada Programming/Libraries/Ada.Streams.Stream_IO[24]
  - Ada Programming/Libraries/Ada.Text_IO.Text_Streams[25]
- Ada Programming/Libraries/Ada.Text_IO[26]
  - Ada Programming/Libraries/Ada.Text_IO.Enumeration_IO[27] (nested package)
  - Ada Programming/Libraries/Ada.Text_IO.Integer_IO[28] (nested package)
  - Ada Programming/Libraries/Ada.Text_IO.Modular_IO[29] (nested package)
  - Ada Programming/Libraries/Ada.Text_IO.Float_IO[30] (nested package)
  - Ada Programming/Libraries/Ada.Text_IO.Fixed_IO[31] (nested package)
  - Ada Programming/Libraries/Ada.Text_IO.Decimal_IO[32] (nested package)
  - Ada Programming/Libraries/Ada.Text_IO.Bounded_IO[33]
  - Ada Programming/Libraries/Ada.Text_IO.Unbounded_IO[34]
  - Ada Programming/Libraries/Ada.Text_IO.Complex_IO[35] (specialized needs annex)
  - Ada Programming/Libraries/Ada.Text_IO.Editing[36] (specialized needs annex)
- Ada Programming/Libraries/Ada.Integer_Text_IO[37]
- Ada Programming/Libraries/Ada.Float_Text_IO[38]
- Ada Programming/Libraries/Ada.Complex_Text_IO[39] (specialized needs annex)
- Ada Programming/Libraries/Ada.Storage_IO[40] (not a general-purpose I/O package)
- Ada Programming/Libraries/Ada.IO_Exceptions[41]
- Ada Programming/Libraries/Ada.Command_Line[42]

20 http://en.wikibooks.org/wiki/Ada%20Programming
21 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Direct_IO
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Sequential_IO
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Streams
24 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Streams.Stream_IO
25 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Text_Streams
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Enumeration_IO
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Integer_IO
29 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Modular_IO
30 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Float_IO
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Fixed_IO
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Decimal_IO
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Bounded_IO
34 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Unbounded_IO
35 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Complex_IO
36 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Editing
37 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Integer_Text_IO
38 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Float_Text_IO
39 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Complex_Text_IO
40 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Storage_IO
41 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.IO_Exceptions
42 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Command_Line

- Ada Programming/Libraries/Ada.Directories[43]
- Ada Programming/Libraries/Ada.Environment_Variables[44]
- Ada Programming/Libraries/GNAT.IO[45] (implementation defined)
- Ada Programming/Libraries/GNAT.IO_Aux[46] (implementation defined)
- Ada Programming/Libraries/GNAT.Calendar.Time_IO[47] (implementation defined)
- Ada Programming/Libraries/System.IO[48] (implementation defined)
- Ada Programming/Libraries[49]
  - Ada Programming/Libraries/GUI[50]
  - Ada Programming/Libraries/Web[51]
  - Ada Programming/Libraries/Database[52]
- Ada Programming/Platform[53]
- Ada Programming/Platform/Linux[54]
- Ada Programming/Platform/Windows[55]

18.6.2 Ada Reference Manual

- A.6 Input-Output ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-6.html}
- A.7 External Files and File Objects ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-7.html}
- A.8 Sequential and Direct Files ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-8.html}
- A.10 Text Input-Output ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-10.html}
- A.11 Wide Text Input-Output and Wide Wide Text Input-Output ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-11.html}
- A.12 Stream Input-Output ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-12.html}
- A.13 Exceptions in Input-Output ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-13.html}
- A.14 File Sharing ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-14.html}

43 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Directories
44 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Environment_Variables
45 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.IO
46 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.IO_Aux
47 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Calendar.Time_IO
48 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FSystem.IO
49 Chapter 40 on page 331
50 Chapter 48 on page 361
51 Chapter 51 on page 371
52 Chapter 50 on page 365
53 Chapter 53 on page 375
54 Chapter 54 on page 377
55 Chapter 55 on page 379

18.6.3 Ada 95 Quality and Style Guide

- 7.7 Input/Output ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-7.html}
  - 7.7.1 Name and Form Parameters ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-7-1.html}
  - 7.7.2 File Closing ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-7-2.html}
  - 7.7.3 Input/Output on Access Types ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-7-3.html}
  - 7.7.4 Package Ada.Streams.Stream_IO ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-7-4.html}
  - 7.7.5 Current Error Files ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-7-5.html}

19 Exceptions

19.1 Robustness

Robustness

Robustness is the ability of a system or system component to behave "reasonably" when it detects an anomaly, e.g.:

- It receives invalid inputs.
- Another system component (hardware or software) malfunctions.

Take as example a telephone exchange control program. What should the control program do when a line fails? It is unacceptable simply to halt — all calls will then fail. Better would be to abandon the current call (only), record that the line is out of service, and continue. Better still would be to try to reuse the line — the fault might be transient. Robustness is desirable in all systems, but it is essential in systems on which human safety or welfare depends, e.g., hospital patient monitoring, aircraft fly-by-wire, nuclear power station control, etc.

19.2 Modules, preconditions and postconditions

Modules, preconditions and postconditions

A module may be specified in terms of its preconditions and postconditions. A *precondition* is a condition that the module's inputs are supposed to satisfy. A *postcondition* is a condition that the module's outputs are required to satisfy, provided that the precondition is satisfied. What should a module do if its precondition is not satisfied?

- Halt? Even with diagnostic information, this is generally unacceptable.
- Use a global result code? The result code can be set to indicate an anomaly. Subsequently it may be tested by a module that can effect error recovery. Problem: this induces tight coupling among the modules concerned.
- Each module has its own result code? This is a parameter (or function result) that may be set to indicate an anomaly, and is tested by calling modules. Problems: (1) setting and testing result codes tends to swamp the normal-case logic and (2) the result codes are normally ignored.
- Exception handling — Ada's solution. A module detecting an anomaly raises an exception. The same, or another, module may handle that exception.

The exception mechanism permits clean, modular handling of anomalous situations:

- A unit (e.g., block or subprogram body) may raise an exception, to signal that an anomaly has been detected. The computation that raised the exception is abandoned (and can never be resumed, although it can be restarted).
- A unit may propagate an exception that has been raised by itself (or propagated out of another unit it has called).
- A unit may alternatively handle such an exception, allowing programmer-defined recovery from an anomalous situation. Exception handlers are segregated from normal-case code.

19.3 Predefined exceptions

Predefined exceptions

The predefined exceptions are those defined in package Standard . Every language-defined run-time error causes a predefined exception to be raised. Some examples are:

- `Constraint_Error`, raised when a subtype's constraint is not satisfied
- `Program_Error`, when a protected operation is called inside a protected object, e.g.
- `Storage_Error`, raised by running out of storage
- `Tasking_Error`, when a task cannot be activated because the operating system has not enough resources, e.g.

Ex.1

```
Name : String (1 .. 10);
...
Name := "Hamlet"; --Raises Constraint_Error,
                  --because the "Hamlet" has bounds (1 ..  6).
```

Ex.2

```
loop
   P := new Int_Node'(0, P);
end loop; --Soon raises Storage_Error,
          --because of the extreme memory leak.
```

Ex.3 Compare the following approaches:

```
procedure Compute_Sqrt (X    : in  Float;
                        Sqrt : out Float;
                        OK   : out Boolean)
is
begin
   if X >= 0 then
      OK := True;
      --compute √X
      ...
   else
      OK := False;
   end if;
end Compute_Sqrt;
```

```
...

procedure Triangle (A, B, C         : in  Float;
                    Area, Perimeter : out Float;
                    Exists          : out Boolean)
is
   S  : Float := 0.5 * (A + B + C);
   OK : Boolean;
begin
   Compute_Sqrt (S * (S-A) * (S-B) * (S-C), Area, OK);
   Perimeter := 2.0 * S;
   Exists    := OK;
end Triangle;
```

A negative argument to Compute_Sqrt causes OK to be set to False. Triangle uses it to determine its own status parameter value, and so on up the calling tree, *ad nauseam*.

versus

```
function Sqrt (X : Float) return Float is
begin
   if X < 0.0 then
      raise Constraint_Error;
   end if;
   --compute √X
   ...
end Sqrt;

...

procedure Triangle (A, B, C         : in  Float;
                    Area, Perimeter : out Float)
is
   S  : Float := 0.5 * (A + B + C);
   OK : Boolean;
begin
   Area      := Sqrt (S * (S-A) * (S-B) * (S-C));
   Perimeter := 2.0 * S;
end Triangle;
```

A negative argument to Sqrt causes Constraint_Error to be explicitly raised inside Sqrt, and propagated out. Triangle simply propagates the exception (by not handling it).

Alternatively, we can catch the error by using the type system:

```
subtype Pos_Float is Float range 0.0 .. Float'Last;

function Sqrt (X : Pos_Float) return Pos_Float is
begin
   --compute √X
   ...
end Sqrt;
```

A negative argument to Sqrt now raises Constraint_Error at the point of call. Sqrt is never even entered.

19.4 Input-output exceptions

Input-output exceptions

Some examples of exceptions raised by subprograms of the **predefined package** Ada.Text_-IO are:

- `End_Error`, raised by Get, Skip_Line, etc., if end-of-file already reached.
- `Data_Error`, raised by Get in Integer_IO, etc., if the input is not a literal of the expected type.
- `Mode_Error`, raised by trying to read from an output file, or write to an input file, etc.
- `Layout_Error`, raised by specifying an invalid data format in a text I/O operation

Ex. 1

```
declare
   A : Matrix (1 .. M, 1 .. N);
begin
   for I in 1 .. M loop
      for J in 1 .. N loop
          begin
             Get (A(I,J));
          exception
             when Data_Error =>
                Put ("Ill-formed matrix element");
                A(I,J) := 0.0;
          end;
       end loop;
   end loop;
exception
   when End_Error =>
      Put ("Matrix element(s) missing");
end;
```

19.5 Exception declarations

Exception declarations

Exceptions are declared rather like objects, but they are not objects. For example, recursive re-entry to a scope where an exception is declared does *not* create a new exception of the same name; instead the exception declared in the outer invocation is reused.

Ex.1

```
Line_Failed : exception;
```

Ex.2

```
package Directory_Enquiries is

   procedure Insert (New_Name   : in Name;
                     New_Number : in Number);

   procedure Lookup (Given_Name  : in  Name;
```

```
                     Corr_Number : out Number);

   Name_Duplicated : exception;
   Name_Absent     : exception;
   Directory_Full  : exception;

end Directory_Enquiries;
```

19.6 Raising exceptions

Raising exceptions

The **raise** statement explicitly raises a specified exception.

Ex. 1

```
package body Directory_Enquiries is

   procedure Insert (New_Name   : in Name;
                     New_Number : in Number)
   is
      ...
   begin
      ...
      if New_Name = Old_Entry.A_Name then
         raise Name_Duplicated;
      end if;
      ...
      New_Entry :=  new Dir_Node'(New_Name, New_Number,...);
      ...
   exception
      when Storage_Error => raise Directory_Full;
   end Insert;

   procedure Lookup (Given_Name  : in  Name;
                     Corr_Number : out Number)
   is
      ...
   begin
      ...
      if not Found then
         raise Name_Absent;
      end if;
      ...
   end Lookup;

end Directory_Enquiries;
```

19.7 Exception handling and propagation

Exception handling and propagation

Exception handlers may be grouped at the end of a block, subprogram body, etc. A handler is any sequence of statements that may end:

- by completing;
- by executing a **return** statement;
- by raising a different exception (**raise** e;);
- by re-raising the same exception (**raise**;).

Suppose that an exception e is raised in a sequence of statements U (a block, subprogram body, etc.).

- If U contains a handler for e: that handler is executed, then control leaves U.
- If U contains no handler for e: e is *propagated* out of U; in effect, e is raised at the "point of call" of U.

So the raising of an exception causes the sequence of statements responsible to be abandoned at the point of occurrence of the exception. It is not, and cannot be, resumed.

Ex. 1

```
...
exception
   when Line_Failed =>
      begin --attempt recovery
         Log_Error;
         Retransmit (Current_Packet);
      exception
         when Line_Failed =>
            Notify_Engineer; --recovery failed!
            Abandon_Call;
      end;
...
```

19.8 Information about an exception occurrence

Information about an exception occurrence

Ada provides information about an exception in an object of type Exception_Occurrence, defined in Ada.Exceptions along with subprograms taking this type as parameter:

- Exception_Name: return the full exception name using the dot notation and in uppercase letters. For example, `Queue.Overflow`.
- Exception_Message: return the exception message associated with the occurrence.
- Exception_Information: return a string including the exception name and the associated exception message.

For getting an exception occurrence object the following syntax is used:

```
with Ada ;  use Ada ;
...
exception
  when Error: High_Pressure | High_Temperature =>
    Put ("Exception: ");
    Put_Line (Exception_Name (Error));
    Put (Exception_Message (Error));
  when Error: others =>
    Put ("Unexpected exception: ");
```

```
    Put_Line (Exception_Information(Error));
end;
```

The exception message content is implementation defined when it is not set by the user who raises the exception. It usually contains a reason for the exception and the raising location.

The user can specify a message using the procedure Raise_Exception.

```
declare
   Valve_Failure : exception;
begin
  ...
  Raise_Exception (Valve_Failure'Identity, "Failure while opening");
  ...
  Raise_Exception (Valve_Failure'Identity, "Failure while closing");
  ...
exception
  when Fail: Valve_Failure =>
    Put (Exception_Message (Fail));
end;
```

Starting with Ada 2005, a simpler syntax can be used to associate a string message with exception occurrence.

```
-- This language feature is only available in Ada 2005
declare
   Valve_Failure : exception;
begin
  ...
  raise Valve_Failure with "Failure while opening";
  ...
  raise Valve_Failure with "Failure while closing";
  ...
exception
  when Fail: Valve_Failure =>
    Put (Exception_Message (Fail));
end;
```

The Ada.Exceptions package also provides subprograms for saving exception occurrences and re-raising them.

19.9 See also

See also

19.9.1 Wikibook

- Ada Programming[1]

1 http://en.wikibooks.org/wiki/Ada%20Programming

19.9.2 Ada 95 Reference Manual

- Section 11: Exceptions ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-11.html}
- 11.4.1 The Package Exceptions ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-11-4-1.html}

19.9.3 Ada 2005 Reference Manual

- Section 11: Exceptions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-11.html}
- 11.4.1 The Package Exceptions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-11-4-1.html}

19.9.4 Ada Quality and Style Guide

- **Chapter 4: Program Structure**
  - 4.3 Exceptions ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_4/4-3.html}
    - 4.3.1 Using Exceptions to Help Define an Abstraction ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_4/4-3-1.html}
- **Chapter 5: Programming Practices**
  - 5.8 Using Exceptions ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-8.html}
    - 5.8.1 Handling Versus Avoiding Exceptions ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-8-1.html}
    - 5.8.2 Handling for Others ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-8-2.html}
    - 5.8.3 Propagation ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-8-3.html}
    - 5.8.4 Localizing the Cause of an Exception ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-8-4.html}
- **Chapter 7: Portability**
  - 7.5 Exceptions ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-5.html}
    - 7.5.1 Predefined and User-Defined Exceptions ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-5-1.html}
    - 7.5.2 Implementation-Specific Exceptions ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-5-2.html}

20 Generics

20.1 Parametric polymorphism (generic units)

Parametric polymorphism (generic units)

The idea of code reuse arises from the necessity for constructing large software systems combining well-established building blocks. The reusability of code improves the productivity and the quality of software. The generic units are one of the ways in which the Ada language supports this characteristic. A generic unit is a subprogram or package that defines algorithms in terms of types and operations that are not defined until the user instantiates them.

Note to C++ programmers: generic units are similar to C++ templates.

For example, to define a procedure for swapping variables of any (non-limited) type:

```
generic
  type Element_T is private;  --Generic formal type parameter
procedure Swap (X, Y : in out Element_T);
```

```
procedure Swap (X, Y : in out Element_T) is
  Temporary : constant Element_T := X;
begin
  X := Y;
  Y := Temporary;
end Swap;
```

The `Swap` subprogram is said to be generic. The subprogram specification is preceded by the generic formal part consisting of the reserved word **generic** followed by a list of generic formal parameters which may be empty. The entities declared as generic are not directly usable, it is necessary to instantiate them.

To be able to use `Swap`, it is necessary to create an instance for the wanted type. For example:

```
procedure Swap_Integers is new Swap (Integer);
```

Now the `Swap_Integers` procedure can be used for variables of type `Integer`.

The generic procedure can be instantiated for all the needed types. It can be instantiated with different names or, if the same identifier is used in the instantiation, each declaration overloads the procedure:

```
procedure Instance_Swap is new Swap (Float);
procedure Instance_Swap is new Swap (Day_T);
procedure Instance_Swap is new Swap (Element_T => Stack_T);
```

Similarly, generic packages can be used, for example, to implement a stack of any kind of elements:

```
generic
  Max: Positive;
  type Element_T is private;
package Generic_Stack is
  procedure Push (E: Element_T);
  function Pop return Element_T;
end Generic_Stack;
```

```
package body Generic_Stack is
  Stack: array (1 .. Max) of Element_T;
  Top  : Integer range 0 .. Max := 0;  --initialise to empty
  --...
end Generic_Stack;
```

A stack of a given size and type could be defined in this way:

```
declare
  package Float_100_Stack is new Generic_Stack (100, Float);
  use Float_100_Stack;
begin
  Push (45.8);
  --...
end;
```

20.2 Generic parameters

Generic parameters

The generic unit declares *generic formal parameters*, which can be:

- objects (of mode *in* or *in out* but never *out*)
- types
- subprograms
- instances of another, designated, generic unit.

When instantiating the generic, the programmer passes one *actual parameter* for each formal. Formal values and subprograms can have defaults, so passing an actual for them is optional.

20.2.1 Generic formal objects

Formal parameters of mode *in* accept any value, constant, or variable of the designated type. The actual is copied into the generic instance, and behaves as a constant inside the generic; this implies that the designated type cannot be limited. It is possible to specify a

default value, like this:

```
generic
   Object : in Natural := 0;
```

For mode *in out*, the actual must be a variable.

One limitation with generic formal objects is that they are never considered static, even if the actual happens to be static. If the object is a number, it cannot be used to create a new type. It can however be used to create a new derived type, or a subtype:

```
generic
   Size : in Natural := 0;
package P is
   type T1 is mod Size; --illegal!
   type T2 is range 1 .. Size; --illegal!
   type T3 is new Integer range 1 .. Size; --OK
   subtype T4 is Integer range 1 .. Size; --OK
end P;
```

The reason why formal objects are nonstatic is to allow the compiler to emit the object code for the generic only once, and to have all instances share it, passing it the address of their actual object as a parameter. This bit of compiler technology is called *shared generics*. If formal objects were static, the compiler would have to emit one copy of the object code, with the object embedded in it, for each instance, potentially leading to an explosion in object code size (*code bloat*).

(Note to C++ programmers: in C++, since formal objects can be static, the compiler cannot implement shared generics in the general case; it would have to examine the entire body of the generic before deciding whether or not to share its object code. In contrast, Ada generics are designed so that the compiler can instantiate a generic *without looking at its body*.)

20.2.2 Generic formal types

The syntax allows the programmer to specify which type categories are acceptable as actuals. As a rule of thumb: The syntax expresses how the generic sees the type, i.e. it assumes the worst, not how the creator of the instance sees the type.

This is the syntax of RM 12.5:

```
 formal_type_declaration ::=
   type defining_identifier[discriminant_part] is
formal_type_definition;

 formal_type_definition ::= formal_private_type_definition
                          | formal_derived_type_definition
                          | formal_discrete_type_definition
                          | formal_signed_integer_type_definition
                          | formal_modular_type_definition
                          | formal_floating_point_definition
                          | formal_ordinary_fixed_point_definition
                          | formal_decimal_fixed_point_definition
```

```
                        | formal_array_type_definition
                        | formal_access_type_definiton
                        | formal_interface_type_definition
```

This is quite complex, so some examples are given below. A type declared with the syntax `type T (<>)` denotes a type with *unknown discriminants*. This is the Ada vernacular for indefinite types, i.e. types for which objects cannot be declared without giving an initial expression. An example of such a type is one with a discriminant without default, another example is an unconstrained array type.

Generic formal type	Acceptable actual types
`type T (<>) is limited private;`	Any type at all. The actual type can be limited[1] or not, indefinite or definite, but the *generic* treats it as limited and indefinite, i.e. does not assume that assignment is available for the type.
`type T (<>) is private;`	Any nonlimited type: the generic knows that it is possible to assign to variables of this type, but it is not possible to declare objects of this type without initial value.
`type T is private;`	Any nonlimited definite type: the generic knows that it is possible to assign to variables of this type and to declare objects without initial value.
`type T (<>) is abstract tagged limited private;`	Any tagged type[2], abstract or concrete, limited or not.
`type T (<>) is tagged limited private;`	Any concrete tagged type, limited or not.
`type T (<>) is abstract tagged private;`	Any nonlimited tagged type, abstract or concrete.
`type T (<>) is tagged private;`	Any nonlimited, concrete tagged type.
`type T (<>) is new Parent;`	Any type derived from `Parent`. The generic knows about `Parent`'s operations, so can call them. Neither `T` nor `Parent` can be abstract.
`type T (<>) is abstract new Parent with private;`	Any type, abstract or concrete, derived from `Parent`, where `Parent` is a tagged type, so calls to `T`'s operations can dispatch dynamically.
`type T (<>) is new Parent with private;`	Any concrete type, derived from the tagged type `Parent`.

1 Chapter 14 on page 113
2 Chapter 22 on page 187

Generic formal type	Acceptable actual types
`type T is (<>);`	Any discrete type: integer[3], modular[4], or enumeration[5].
`type T is range <>;`	Any signed integer type
`type T is mod <>;`	Any modular type
`type T is delta <>;`	Any (non-decimal) fixed point type[6]
`type T is delta <> digits <>;`	Any decimal fixed point type
`type T is digits <>;`	Any floating point type[7]
type T **is array** (I) **of** E;	Any array type[8] with index of type `I` and elements of type `E` (`I` and `E` could be formal parameters as well)
`type T is access O;`	Any access type[9] pointing to objects of type `O` (`O` could be a formal parameter as well)

In the body we can only use the operations predefined for the type category of the formal parameter. That is, the generic specification is a contract between the generic implementor and the client instantiating the generic unit. This is different to the parametric features of other languages, such as C++.

It is possible to further restrict the set of acceptable actual types like so:

Generic formal type	Acceptable actual types
`type T (<>) is`...	Definite or indefinite types (loosely speaking: types with or without discriminants, but other forms of indefiniteness exist)
`type T (D : DT) is`...	Types with a discriminant of type DT (it is possible to specify several discriminants, too)
`type T is`...	Definite types (loosely speaking types without a discriminant or with a discriminant with default value)

20.2.3 Generic formal subprograms

It is possible to pass a subprogram as a parameter to a generic. The generic specifies a generic formal subprogram, complete with parameter list and return type (if the subprogram is a function). The actual must match this parameter profile. It is not necessary that the *names* of parameters match, though.

3 Chapter 6 on page 69
4 Chapter 7 on page 71
5 Chapter 8 on page 73
6 Chapter 10 on page 79
7 Chapter 9 on page 77
8 Chapter 11 on page 83
9 Chapter 13 on page 99

Here is the specification of a generic subprogram that takes another subprogram as its parameter:

```
generic
  type Element_T is private;
  with function "*" (X, Y: Element_T) return Element_T;
function Square (X : Element_T) return Element_T;
```

And here is the body of the generic subprogram; it calls parameter as it would any other subprogram.

```
function Square (X: Element_T) return Element_T is
begin
  return X * X;   --The formal operator "*".
end Square;
```

This generic function could be used, for example, with matrices, having defined the matrix product.

```
with Square;
with Matrices;
procedure Matrix_Example is
  function Square_Matrix is new Square
    (Element_T => Matrices.Matrix_T, "*" => Matrices.Product);
  A : Matrices.Matrix_T := Matrices.Identity;
begin
  A := Square_Matrix (A);
end Matrix_Example;
```

It is possible to specify a default with "the box" (`is <>`), like this:

```
generic
  type Element_T is private;
  with function "*" (X, Y: Element_T) return Element_T is <>;
```

This means that if, at the point of instantiation, a function "*" exists for the actual type, and if it is directly visible, then it will be used by default as the actual subprogram.

One of the main uses is passing needed operators. The following example shows this (follow download links for full example):

File: Algorithms/binary_search.adb

```
generic
   type Element_Type is private;
   ...
   with function "<"
     (Left  : in Element_Type;
      Right : in Element_Type)
      return  Boolean
```

```
      is <> ;
   procedure Search
     (Elements : in Array_Type;
      Search   : in Element_Type;
      Found    : out Boolean;
      Index    : out Index_Type'Base)
      ...
```

20.2.4 Generic instances of other generic packages

A generic formal can be a package; it must be an instance of a generic package, so that the generic knows the interface exported by the package:

```
generic
   with package P is new Q (<> );
```

This means that the actual must be an instance of the generic package Q. The box after Q means that we do not care which actual generic parameters were used to create the actual for P. It is possible to specify the exact parameters, or to specify that the defaults must be used, like this:

```
generic
   --P1 must be an instance of Q with the specified actual parameters:
   with package P1 is new Q (Param1 => X, Param2 => Y);

   --P2 must be an instance of Q where the actuals are the defaults:
   with package P2 is new Q;
```

It is all or nothing: if you specify the generic parameters, you must specify all of them. Similarly, if you specify no parameters and no box, then all the generic formal parameters of Q must have defaults. The actual package must, of course, match these constraints.

The generic sees both the public part and the generic parameters of the actual package (Param1 and Param2 in the above example).

This feature allows the programmer to pass arbitrarily complex types as parameters to a generic unit, while retaining complete type safety and encapsulation. *(example needed)*

It is not possible for a package to list itself as a generic formal, so no generic recursion is possible. The following is illegal:

```
with A;
generic
   with package P is new A (<> );
package A; --illegal:  A references itself
```

In fact, this is only a particular case of:

```
with A; --illegal:  A does not exist yet at this point!
package A;
```

which is also illegal, despite the fact that A is no longer generic.

20.3 Instantiating generics

Instantiating generics

To instantiate a generic unit, use the keyword **new**:

```
function Square_Matrix is new Square
   (Element_T => Matrices.Matrix_T, "*" => Matrices.Product);
```

Notes of special interest to C++ programmers:

- The generic formal types define *completely* which types are acceptable as actuals; therefore, the compiler can instantiate generics without looking at the body of the generic.
- Each instance has a name and is different from all other instances. In particular, if a generic package declares a type, and you create two instances of the package, then you will get two different, incompatible types, even if the actual parameters are the same.
- Ada requires that all instantiations be explicit.
- It is not possible to create special-case instances of a generic (known as "template specialisation" in C++).

As a consequence of the above, Ada does not permit template metaprogramming. However, this design has significant advantages:

- the object code can be shared by all instances of a generic, unless of course the programmer has requested that subprograms be inlined; there is no danger of code bloat.
- when reading programs written by other people, there are no hidden instantiations, and no special cases to worry about. Ada follows the Law of Least Astonishment.

20.4 Advanced generics

Advanced generics

20.4.1 Generics and nesting

A generic unit can be nested inside another unit, which itself may be generic. Even though no special rules apply (just the normal rules about generics and the rules about nested units), novices may be confused. It is important to understand the difference between a generic unit and *instances* of a generic unit.

Example 1. A generic subprogram nested in a nongeneric package.

```
package Bag_Of_Strings is
   type Bag is private;
   generic
      with procedure Operator (S : in out String);
   procedure Apply_To_All (B : in out Bag);
private
   --omitted
end Bag_Of_Strings;
```

To use **Apply_To_All**, you first define the procedure to be applied to each String in the Bag. Then, you instantiate **Apply_To_All**, and finally you call the instance.

```
with Bag_Of_Strings;
procedure Example_1 is
   procedure Capitalize (S : in out String) is separate; --omitted
   procedure Capitalize_All is
      new Bag_Of_Strings.Apply_To_All (Operator => Capitalize);
   B : Bag_Of_Strings.Bag;
begin
   Capitalize_All (B);
end Example_1;
```

Example 2. A generic subprogram nested in a generic package

This is the same as above, except that now the Bag itself is generic:

```
generic
   type Element_Type (<>) is private;
package Generic_Bag is
   type Bag is private;
   generic
      with procedure Operator (S : in out Element_Type);
   procedure Apply_To_All (B : in out Bag);
private
   --omitted
end Generic_Bag;
```

As you can see, the generic formal subprogram **Operator** takes a parameter of the generic formal type **Element_Type**. This is okay: the nested generic sees everything that is in its enclosing unit.

You cannot instantiate **Generic_Bag.Apply_To_All** directly, so you must first create an instance of **Generic_Bag**, say **Bag_Of_Strings**, and then instantiate **Bag_Of_Strings.Apply_To_All**.

```
with Generic_Bag;
procedure Example_2 is
   procedure Capitalize (S : in out String) is separate; --omitted
   package Bag_Of_Strings is
      new Generic_Bag (Element_Type => String);
   procedure Capitalize_All is
      new Bag_Of_Strings.Apply_To_All (Operator => Capitalize);
   B : Bag_Of_Strings.Bag;
begin
```

```
   Capitalize_All (B);
end Example_2;
```

20.4.2 Generics and child units

Example 3. A generic unit that is a child of a nongeneric unit.

Each instance of the generic child is a child of the parent unit, and so it can see the parent's public and private parts.

```
package Bag_Of_Strings is
   type Bag is private;
private
   --omitted
end Bag_Of_Strings;

generic
   with procedure Operator (S : in out String);
procedure Bag_Of_Strings.Apply_To_All (B : in out Bag);
```

The differences between this and Example 1 are:

- **Bag__Of__Strings.Apply__To__All** is compiled separately. In particular, **Bag__Of__Strings.Apply__To__All** might have been written by a different person who did not have access to the source text of **Bag__Of__Strings**.
- Before you can use **Bag__Of__Strings.Apply__To__All**, you must **with** it explicitly; **with**ing the parent, **Bag__Of__Strings**, is not sufficient.
- If you do not use **Bag__Of__Strings.Apply__To__All**, your program does not contain its object code.
- Because **Bag__Of__Strings.Apply__To__All** is at the library level, it can declare controlled types; the nested package could not do that in Ada 95. In Ada 2005, one can declare controlled types at any level.

```
with Bag_Of_Strings.Apply_To_All; --implicitly withs Bag_Of_Strings, too
procedure Example_3 is
   procedure Capitalize (S : in out String) is separate; --omitted
   procedure Capitalize_All is
      new Bag_Of_Strings.Apply_To_All (Operator => Capitalize);
   B : Bag_Of_Strings.Bag;
begin
   Capitalize_All (B);
end Example_3;
```

Example 4. A generic unit that is a child of a generic unit

This is the same as Example 3, except that now the Bag is generic, too.

```
generic
   type Element_Type (<>) is private;
package Generic_Bag is
   type Bag is private;
private
```

```
   --omitted
end Generic_Bag;

generic
   with procedure Operator (S : in out Element_Type);
procedure Generic_Bag.Apply_To_All (B : in out Bag);

with Generic_Bag.Apply_To_All;
procedure Example_4 is
   procedure Capitalize (S : in out String) is separate; --omitted
   package Bag_Of_Strings is
      new Generic_Bag (Element_Type => String);
   procedure Capitalize_All is
      new Bag_Of_Strings.Apply_To_All (Operator => Capitalize);
   B : Bag_Of_Strings.Bag;
begin
   Capitalize_All (B);
end Example_4;
```

Example 5. A parameterless generic child unit

Children of a generic unit **must** be generic, no matter what. If you think about it, it is quite logical: a child unit sees the public and private parts of its parent, including the variables declared in the parent. If the parent is generic, which instance should the child see? The answer is that the child must be the child of only one instance of the parent, therefore the child must also be generic.

```
generic
   type Element_Type (<>) is private;
   type Hash_Type is (<>);
   with function Hash_Function (E : Element_Type) return Hash_Type;
package Generic_Hash_Map is
   type Map is private;
private
   --omitted
end Generic_Hash_Map;
```

Suppose we want a child of a **Generic_Hash_Map** that can serialise the map to disk; for this it needs to sort the map by hash value. This is easy to do, because we know that **Hash_Type** is a discrete type, and so has a less-than operator. The child unit that does the serialisation does not need any additional generic parameters, but it must be generic nevertheless, so it can see its parent's generic parameters, public and private part.

```
generic
package Generic_Hash_Map.Serializer is
    procedure Dump (Item : in Map; To_File : in String);
    procedure Restore (Item : out Map; From_File : in String);
end Generic_Hash_Map.Serializer;
```

To read and write a map to disk, you first create an instance of **Generic_Hash_Map**, for example **Map_Of_Unbounded_Strings**, and then an instance of **Map_Of_-Unbounded_Strings.Serializer**:

```
with Ada.Strings.Unbounded;
with Generic_Hash_Map.Serializer;
procedure Example_5 is
   use Ada.Strings.Unbounded;
   function Hash (S : in Unbounded_String) return Integer is separate; --omitted
   package Map_Of_Unbounded_Strings is
      new Generic_Hash_Map (Element_Type => Unbounded_String,
                            Hash_Type => Integer,
                            Hash_Function => Hash);
   package Serializer is
      new Map_Of_Unbounded_Strings.Serializer;
   M : Map_Of_Unbounded_Strings.Map;
begin
   Serializer.Restore (Item => M, From_File => "map.dat");
end Example_5;
```

20.5 See also

See also

20.5.1 Wikibook

- Ada Programming[10]
- Ada Programming/Object Orientation[11]: tagged types provides other mean of polymorphism in Ada.

20.5.2 Wikipedia

- Generic programming[12]

20.5.3 Ada Reference Manual

- Section 12: Generic Units ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-12.html}

es:Programación en Ada/Unidades genéricas[13]

10 http://en.wikibooks.org/wiki/Ada%20Programming
11 Chapter 22 on page 187
12 http://en.wikipedia.org/wiki/Generic%20programming
13 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FUnidades%20gen%E9ricas

21 Tasking

21.1 Tasks

Tasks

A *task unit* is a program unit that is obeyed concurrently with the rest of an Ada program. The corresponding activity, a new locus of control, is called a *task* in Ada terminology, and is similar to a *thread*, for example in Java Threads[1]. The execution of the main program is also a task, the anonymous environment task. A task unit has both a declaration and a body, which is mandatory. A task body may be compiled separately as a subunit, but a task may not be a library unit, nor may it be generic. Every task depends on a *master*, which is the immediately surrounding declarative region - a block, a subprogram, another task, or a package. The execution of a master does not complete until all its dependent tasks have terminated. The environment task is the master of all other tasks; it terminates only when all other tasks have terminated.

Task units are similar to packages in that a task declaration defines entities exported from the task, whereas its body contains local declarations and statements of the task.

A single task is declared as follows:

```
task Single is
   declarations of exported identifiers
end Single;
...
task body Single is
  local declarations and statements
end Single;
```

A task declaration can be simplified, if nothing is exported, thus:

```
task No_Exports;
```

Ex. 1

```
procedure Housekeeping is

   task Check_CPU;
   task Backup_Disk;
```

1 http://en.wikibooks.org/wiki/Java%3AThreads

```
   task body Check_CPU is
      ...
   end Check_CPU;

   task body Backup_Disk is
      ...
   end Backup_Disk;
   --the two tasks are automatically created and begin execution
begin --Housekeeping
   null;
   --Housekeeping waits here for them to terminate
end Housekeeping;
```

It is possible to declare task types, thus allowing task units to be created dynamically, and incorporated in data structures:

```
task type T is
   ...
end T;
...
Task_1, Task_2 : T;
...
task body T is
   ...
end T;
```

Task types are **limited**, i.e. they are restricted in the same way as limited private types, so assignment and comparison are not allowed.

21.1.1 Rendezvous

The only entities that a task may export are entries. An **entry** looks much like a procedure. It has an identifier and may have **in**, **out** or **in out** parameters. Ada supports communication from task to task by means of the *entry call*. Information passes between tasks through the actual parameters of the entry call. We can encapsulate data structures within tasks and operate on them by means of entry calls, in a way analogous to the use of packages for encapsulating variables. The main difference is that an entry is executed by the called task, not the calling task, which is suspended until the call completes. If the called task is not ready to service a call on an entry, the calling task waits in a (FIFO) queue associated with the entry. This interaction between calling task and called task is known as a *rendezvous*. The calling task requests rendezvous with a specific named task by calling one of its entries. A task accepts rendezvous with any caller of a specific entry by executing an **accept** statement for the entry. If no caller is waiting, it is held up. Thus entry call and accept statement behave symmetrically. (To be honest, optimized object code may reduce the number of context switches below the number implied by this naive description.)

Ex. 2 The following task type implements a single-slot buffer, i.e. an encapsulated variable that can have values inserted and removed in strict alternation. Note that the buffer task has no need of state variables to implement the buffer protocol: the alternation of insertion and removal operations is directly enforced by the control structure in the body of Encapsulated_Buffer_Task_Type which is, as is typical, a **loop**.

```
task type Encapsulated_Buffer_Task_Type is
   entry Insert (An_Item : in  Item);
   entry Remove (An_Item : out Item);
end Encapsulated_Buffer_Task_Type;
...
Buffer_Pool : array (0 .. 15) of Encapsulated_Buffer_Task_Type;
This_Item   : Item;
...
task body Encapsulated_Buffer_Task_Type is
   Datum : Item;
begin
   loop
      accept Insert (An_Item : in  Item) do
         Datum := An_Item;
      end Insert;
      accept Remove (An_Item : out Item) do
         An_Item := Datum;
      end Remove;
   end loop;
end Encapsulated_Buffer_Task_Type;
...
Buffer_Pool(1).Remove (This_Item);
Buffer_Pool(2).Insert (This_Item);
```

21.1.2 Selective Wait

To avoid being held up when it could be doing productive work, a server task often needs the freedom to accept a call on any one of a number of alternative entries. It does this by means of the *selective wait* statement, which allows a task to wait for a call on any of two or more entries.

If only one of the alternatives in a selective wait statement has a pending entry call, then that one is accepted. If two or more alternatives have calls pending, the implementation is free to accept any one of them. For example, it could choose one at random. This introduces *bounded non-determinism* into the program. A sound Ada program should not depend on a particular method being used to choose between pending entry calls. (However, there are facilities to influence the method used, when that is necessary.)

Ex. 3

```
task type Encapsulated_Variable_Task_Type is
   entry Store (An_Item : in  Item);
   entry Fetch (An_Item : out Item);
end Encapsulated_Variable_Task_Type;
...
task body Encapsulated_Variable_Task_Type is
   Datum : Item;
begin
   accept Store (An_Item : in Item) do
      Datum := An_Item;
   end Store;
   loop
      select
         accept Store (An_Item : in Item) do
            Datum := An_Item;
         end Store;
      or
```

```
         accept Fetch (An_Item : out Item) do
            An_Item := Datum;
         end Fetch;
      end select;
   end loop;
end Encapsulated_Variable_Task_Type;
```

```
x, y : Encapsulated_Variable_Task_Type;
```

creates two variables of type Encapsulated_Variable_Task_Type. They can be used thus:

```
it : Item;
...
x.Store(Some_Expression);
...
x.Fetch (it);
y.Store (it);
```

Again, note that the control structure of the body ensures that an Encapsulated_Variable_-Task_Type must be given an initial value by a first Store operation before any Fetch operation can be accepted.

21.1.3 Guards

Depending on circumstances, a server task may not be able to accept calls for some of the entries that have accept alternatives in a selective wait statement. The acceptance of any alternative can be made conditional by using a *guard*, which is *Boolean*[2] precondition for acceptance. This makes it easy to write monitor-like server tasks, with no need for an explicit signaling mechanism, nor for mutual exclusion. An alternative with a True guard is said to be *open*. It is an error if no alternative is open when the selective wait statement is executed, and this raises the Program_Error exception.

Ex. 4

```
task Cyclic_Buffer_Task_Type is
   entry Insert (An_Item : in  Item);
   entry Remove (An_Item : out Item);
end Cyclic_Buffer_Task_Type;
...
task body Cyclic_Buffer_Task_Type is
   Q_Size : constant := 100;
   subtype Q_Range is Positive range 1 .. Q_Size;
   Length : Natural range 0 .. Q_Size := 0;
   Head, Tail : Q_Range := 1;
   Data : array (Q_Range) of Item;
begin
   loop
      select
         when Length < Q_Size =>
            accept Insert (An_Item : in  Item) do
```

2 http://en.wikibooks.org/wiki/Ada%20Programming%2FTypes%23Boolean

```
               Data(Tail) := An_Item;
            end Insert;
            Tail := Tail mod Q_Size + 1;
            Length := Length + 1;
      or
         when Length > 0 =>
            accept Remove (An_Item : out Item) do
               An_Item := Data(Head);
            end Remove;
            Head := Head mod Q_Size + 1;
            Length := Length - 1;
      end select;
   end loop;
end Cyclic_Buffer_Task_Type;
```

21.2 Protected types

Protected types

Tasks allow for encapsulation and safe usage of variable data without the need for any explicit mutual exclusion and signaling mechanisms. Ex. 4 shows how easy it is to write server tasks that safely manage locally-declared data on behalf of multiple clients. There is no need for mutual exclusion of access to the managed data, *because it is never accessed concurrently.* However, the overhead of creating a task merely to serve up some data may be excessive. For such applications, Ada 95 provides **protected** modules. A protected module encapsulates a data structure and exports subprograms that operate on it under automatic mutual exclusion. It also provides automatic, implicit signaling of conditions between client tasks. Again, a protected module can be either a single protected object or a protected type, allowing many protected objects to be created.

A protected module can export only procedures, functions and entries, and its body may contain only the bodies of procedures, functions and entries. The protected data is declared after **private** in its specification, but is accessible only within the protected module's body. Protected procedures and entries may read and/or write its encapsulated data, and automatically exclude each other. Protected functions may only read the encapsulated data, so that multiple protected function calls can be concurrently executed in the same protected object, with complete safety; but protected procedure calls and entry calls exclude protected function calls, and vice versa. Exported entries and subprograms of a protected object are executed by its calling task, as a protected object has no independent locus of control. (To be honest, optimized object code may reduce the number of context switches below the number implied by this naive description.)

Like a task entry, a protected entry can employ a guard to control admission. This provides automatic signaling, and ensures that when a protected entry call is accepted, its guard condition is True, so that a guard provides a reliable precondition for the entry body.

Ex. 5 The following is a simple protected type analogous to the Encapsulated_Buffer task in Ex. 2.

```
protected type Protected_Buffer_Type is
   entry Insert (An_Item : in  Item);
   entry Remove (An_Item : out Item);
private
   Buffer : Item;
   Empty  : Boolean := True;
end Protected_Buffer_Type;
...
protected body Protected_Buffer_Type is
   entry Insert (An_Item : in  Item)
      when Empty is
   begin
      Buffer := An_Item;
      Empty := False;
   end Insert;
   entry Remove (An_Item : out Item)
      when not Empty is
   begin
      An_Item := Buffer;
      Empty := True;
   end Remove;
end Protected_Buffer_Type;
```

Note how the guards, using the state variable Empty, ensure that messages are alternately inserted and removed, and that no attempt can be made to take data from an empty buffer. All this is achieved without explicit signaling or mutual exclusion constructs, whether in the calling task or in the protected type itself.

The notation for calling a protected entry or procedure is exactly the same as that for calling a task entry. This makes it easy to replace one implementation of the abstract type by the other, the calling code being unaffected.

Ex. 6 The following task type implements Dijkstra's semaphore ADT, with FIFO scheduling of resumed processes. The algorithm will accept calls to both Wait and Signal, so long as the semaphore invariant would not be violated. When that circumstance approaches, calls to Wait are ignored for the time being.

```
task type Semaphore_Task_Type is
   entry Initialize (N : in Natural);
   entry Wait;
   entry Signal;
end Semaphore_Task_Type;
...
task body Semaphore_Task_Type is
   Count : Natural;
begin
   accept Initialize (N : in Natural) do
      Count := N;
   end Initialize;
   loop
      select
         when Count > 0 =>
             accept Wait do
                Count := Count - 1;
             end Wait;
      or
             accept Signal;
             Count := Count + 1;
      end select;
```

```
   end loop;
end Semaphore_Task_Type;
```

This task could be used as follows:

```
nr_Full, nr_Free : Semaphore_Task_Type;
...
nr_Full.Initialize (0); nr_Free.Initialize (nr_Slots);
...
nr_Free.Wait; nr_Full.Signal;
```

Alternatively, semaphore functionality can be provided by a protected object, with major efficiency gains.

Ex. 7 The Initialize and Signal operations of this protected type are unconditional, so they are implemented as protected procedures, but the Wait operation must be guarded and is therefore implemented as an entry.

```
protected type Semaphore_Protected_Type is
   procedure Initialize (N : in Natural);
   entry Wait;
   procedure Signal;
private
   Count : Natural := 0;
end Semaphore_Protected_Type;
...
protected body Semaphore_Protected_Type is
   procedure Initialize (N : in Natural) is
   begin
      Count := N;
   end Initialize;
   entry Wait
      when Count > 0 is
   begin
      Count := Count - 1;
   end Wait;
   procedure Signal is
   begin
      Count := Count + 1;
   end Signal;
end Semaphore_Protected_Type;
```

Unlike the task type above, this does not ensure that Initialize is called before Wait or Signal, and Count is given a default initial value instead. Restoring this defensive feature of the task version is left as an exercise for the reader.

21.3 Entry families

Entry families

Sometimes we need a group of related entries. Entry *families*, indexed by a *discrete type*[3], meet this need.

Ex. 8 This task provides a pool of several buffers.

```
type Buffer_Id is Integer range 1 .. nr_Bufs;
...
task Buffer_Pool_Task is
   entry Insert (Buffer_Id) (An_Item : in Item);
   entry Remove (Buffer_Id) (An_Item : out Item);
end Buffer_Pool_Task;
...
task body Buffer_Pool_Task is
   Data   : array (Buffer_Id) of Item;
   Filled : array (Buffer_Id) of Boolean  := (others => False);
begin
  loop
    for I in Data'Range loop
      select
        when not Filled(I) =>
           accept Insert (I) (An_Item : in Item) do
              Data(I) := An_Item;
           end Insert;
           Filled(I) := True;
      or
        when Filled(I) =>
           accept Remove (I) (An_Item : out Item) do
                  An_Item := Data(I);
           end Remove;
           Filled(I) := False;
      else
        null; --N.B. "polling" or "busy waiting"
      end select;
    end loop;
  end loop;
end Buffer_Pool_Task;
...
Buffer_Pool_Task.Remove(K)(This_Item);
```

Note that the busy wait **else null** is necessary here to prevent the task from being suspended on some buffer when there was no call pending for it, because such suspension would delay serving requests for all the other buffers (perhaps indefinitely).

21.4 Termination

Termination

Server tasks often contain infinite loops to allow them to service an arbitrary number of calls in succession. But control cannot leave a task's master until the task terminates, so we need a way for a server to know when it should terminate. This is done by a *terminate alternative* in a selective wait.

3 http://en.wikibooks.org/wiki/ada%20Programming%2FTypes%23List%20of%20Types

Ex. 9

```
task type Terminating_Buffer_Task_Type is
   entry Insert (An_Item : in  Item);
   entry Remove (An_Item : out Item);
end Terminating_Buffer_Task_Type;
...
task body Terminating_Buffer_Task_Type is
   Datum : Item;
begin
   loop
      select
         accept Insert (An_Item : in  Item) do
            Datum := An_Item;
         end Insert;
      or
         terminate;
      end select;
      select
         accept Remove (An_Item : out Item) do
            An_Item := Datum;
         end Remove;
      or
         terminate;
      end select;
   end loop;
end Terminating_Buffer_Task_Type;
```

The task terminates when:

1. at least one terminate alternative is open, and
2. there are no pending calls to its entries, and
3. all other tasks of the same master are in the same state (or already terminated), and
4. the task's master has completed (i.e. has run out of statements to execute).

Conditions (1) and (2) ensure that the task is in a fit state to stop. Conditions (3) and (4) ensure that stopping cannot have an adverse effect on the rest of the program, because no further calls that might change its state are possible.

21.5 Timeout

Timeout

A task may need to avoid being held up by calling to a slow server. A *timed entry call* lets a client specify a maximum delay before achieving rendezvous, failing which the attempted entry call is withdrawn and an alternative sequence of statements is executed.

Ex. 10

```
task Password_Server is
   entry Check (User, Pass : in String; Valid : out Boolean);
   entry Set (User, Pass : in  String);
end Password_Server;
...
User_Name, Password : String (1 .. 8);
...
```

```
Put ("Please give your new password:");
Get_Line (Password);
select
   Password_Server.Set (User_Name, Password);
   Put_Line ("Done");
or
   delay 10.0;
   Put_Line ("The system is busy now, please try again later.");
end select;
```

To time out the *functionality* provided by a task, two distinct entries are needed: one to pass in arguments, and one to collect the result. Timing out on rendezvous with the latter achieves the desired effect.

Ex. 11

```
task Process_Data is
   entry Input (D  : in  Datum);
   entry Output (D  : out Datum);
end Process_Data;

Input_Data, Output_Data : Datum;

loop
   collect Input_Data from sensors;
   Process_Data.Input (Input_Data);
   select
      Process_Data.Output (Output_Data);
      pass Output_Data to display task;
   or
      delay 0.1;
      Log_Error ("Processing did not complete quickly enough.");
   end select;
end loop;
```

Symmetrically, a delay alternative in a selective wait statement allows a server task to withdraw an offer to accept calls after a maximum delay in achieving rendezvous with any client.

Ex. 12

```
task Resource_Lender is
   entry Get_Loan (Period : in Duration);
   entry Give_Back;
end Resource_Lender;
...
task body Resource_Lender is
   Period_Of_Loan : Duration;
begin
   loop
      select
         accept Get_Loan (Period : in Duration) do
            Period_Of_Loan := Period;
         end Get_Loan;
         select
            accept Give_Back;
         or
            delay Period_Of_Loan;
            Log_Error ("Borrower did not give up loan soon
```

```
enough.");
            end select;
         or
            terminate;
         end select;
     end loop;
  end Resource_Lender;
```

21.6 Conditional entry calls

Conditional entry calls

An entry call can be made conditional, so that it is withdrawn if the rendezvous is not immediately achieved. This uses the select statement notation with an **else** part. Thus the constructs

```
select
  Callee.Rendezvous;
else
  Do_something_else;
end select;
```

and

```
select
  Callee.Rendezvous;
or
  delay 0.0;
  Do_something_else;
end select;
```

seem to be conceptually equivalent. However, the attempt to start the rendezvous may take some time, especially if the callee is on another processor, so the *delay 0.0;* may expire although the callee would be able to accept the rendezvous, whereas the *else* construct is safe.

21.7 Requeue statements

Requeue statements

A requeue statement allows an accept statement or entry body to be completed while redirecting to a different or the same entry queue. The called entry has to share the same parameter list or be parameter-less.

21.8 Scheduling

Scheduling

FIFO, priority, priority inversion avoidance, ... to be completed

21.9 Interfaces

Interfaces

This language feature is only available in Ada 2005

Task and Protected types can also implement interfaces[4].

```
type Printable is task interface;

procedure Input (D  : in  Printable);


task Process_Data is new Printable with
   entry Input  (D  : in  Datum);
   entry Output (D  : out Datum);
end Process_Data;
```

21.10 See also

See also

21.10.1 Wikibook

- Ada Programming[5]
- Ada Programming/Libraries/Ada.Storage_IO[6]

21.10.2 Ada Reference Manual

Ada 95

- Section 9: Tasks and Synchronization ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-9.html}

Ada 2005

- 3.9.4 Interface Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-9-4.html}

4 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Finterface
5 http://en.wikibooks.org/wiki/Ada%20Programming
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Storage_IO

- Section 9: Tasks and Synchronization ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-9.html}

21.11 Ada Quality and Style Guide

Ada Quality and Style Guide

- **Chapter 4: Program Structure**
  - 4.1.9 Tasks ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_4/4-1-9.html}
  - 4.1.10 Protected Types ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_4/4-1-10.html}
- Chapter 6: Concurrency[7]

es:Programación en Ada/Tareas[8]

7 http://www.adaic.org/docs/95style/html/sec_6/toc.html
8 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FTareas

22 Object Orientation

22.1 Object orientation in Ada

Object orientation in Ada

Object oriented programming consists in building the software in terms of "objects". An "object" contains data and has a behavior. The data, normally, consists in constants and variables as seen in the rest of this book but could also, conceivably, reside outside the program entirely, i.e. on disk or on the network. The behavior consists in subprograms that operate on the data. What makes Object Orientation unique, compared to procedural programming, is not a single feature but the combination of several features:

- *encapsulation*, i.e. the ability to separate the implementation of an object from its interface; this in turn separates "clients" of the object, who can only use the object in certain predefined ways, from the internals of the object, which have no knowledge of the outside clients.
- *inheritance*, the ability for one type of objects to inherit the data and behavior (subprograms) of another, without necessarily needing to break encapsulation;
- *type extension*, the ability for an object to add new data components and new subprograms on top of the inherited ones and to *replace* some inherited subprograms with its own versions; this is called *overriding*.
- *polymorphism*, the ability for a "client" to use the services of an object without knowing the exact type of the object, i.e. in an abstract way. The actual type of the object can indeed change at run time from one invocation to the next.

It is possible to do object-oriented programming in any language, even assembly. However, type extension and polymorphism are very difficult to get right without language support.

In Ada, each of these concepts has a matching construct; this is why Ada supports object-oriented programming directly.

- Packages provide encapsulation;
- Derived types provide inheritance;
- Record extensions, described below, provide for type extension;
- Class-wide types, also described below, provide for polymorphism.

Ada has had encapsulation and derived types since the first version (MIL-STD-1815 in 1980), which led some to qualify the language as "object-oriented" in a very narrow sense. Record extensions and class-wide types were added in Ada 95. Ada 2005 further adds interfaces. The rest of this chapter covers these aspects.

22.1.1 The simplest object: the Singleton

```
package Directory is
  function Present (Name_Pattern: String) return Boolean;
  generic
     with procedure Visit (Full_Name, Phone_Number, Address: String;
                           Stop: out Boolean);
  procedure Iterate (Name_Pattern: String);
end Directory;
```

The Directory is an object consisting of data (the telephone numbers and addresses, presumably held in an external file or database) and behavior (it can look an entry up and traverse all the entries matching a Name_Pattern, calling Visit on each).

A simple package provides for encapsulation (the inner workings of the directory are hidden) and a pair of subprograms provide the behavior.

This pattern is appropriate when only one object of a certain type must exist; there is, therefore, no need for type extension or polymorphism.

22.1.2 Primitive operations

For the following, we need the definition of primitive operations:

The set of *primitive operations of a type T* consists of those subprograms that:

- are declared immediately within the same package as the type (not within a nested package nor a child package);
- take a parameter of the type or, for functions, *return* an object of the type;
- take an access parameter of the type or, for functions, *return* an access value of the type.

(Also predefined operators like equality "=" are primitive operations.)

An operation can be primitive on two or more types, but only on one tagged type. The following example would be illegal if also B were tagged.

```
package P is
   type A is tagged private;
   type B is private;
   procedure Proc (This: A; That: B); --primitive on A and B
end P;
```

22.1.3 Derived types

Type derivation has been part of Ada since the very start.

```
package P is
  type T is private;
  function Create (Data: Boolean) return T;  --primitive
  procedure Work (Object : in out T);        --primitive
  procedure Work (Pointer: access T);        --primitive
```

```
   type Acc_T is access T;
   procedure Proc (Pointer: Acc_T);           --not primitive
private
   type T is record
     Data: Boolean;
   end record;
end P;
```

The above example creates a type T that contains data (here just a Boolean but it could be anything) and behavior consisting of some subprograms. It also demonstrates encapsulation by placing the details of the type T in the private part of the package.

The primitive operations of T are the function Create, the overloaded procedures Work, and the predefined "=" operator; Proc is not primitive, since it has an *access type* on T as parameter — don't confuse this with an *access parameter*, as used in the second procedure Work. When deriving from T, the primitive operations are inherited.

```
with P;
package Q is
   type Derived is new P.T;
end Q;
```

The type Q.Derived has the same data *and the same behavior* as P.T; it inherits both the data *and the subprograms*. Thus it is possible to write:

```
with Q;
procedure Main is
   Object: Q.Derived := Q.Create (Data => False);
begin
   Q.Work (Object);
end Main;
```

Admittedly, the reasons for writing this may seem obscure. The purpose of this kind of code is to have objects of types P.T and Q.Derived, which are not compatible:

```
Ob1: P.T;
Ob2: Q.Derived;

Ob1 := Ob2;        -- illegal
Ob1 := P.T (Ob2);  -- but convertible
```

This feature is not used very often (it's used e.g. for declaring types reflecting physical dimensions) but I present it here to introduce the next step: type extension.

22.1.4 Type extensions

Type extensions are an Ada 95 amendment.

A tagged type provides support for dynamic polymorphism and type extension. A tagged type bears a hidden tag that identifies the type at run-time. Apart from the tag, a tagged

record is like any other record, so it can contain arbitrary data.

```
package Person is
   type Object is tagged
     record
         Name   : String (1 .. 10);
         Gender : Gender_Type;
     end record;
   procedure Put (O : Object);
end Person;
```

As you can see, a `Person.Object` is an *object* in the sense that it has data and behavior (the procedure `Put`). However, this object does not hide its data; any program unit that has a **`with`** `Person` clause can read and write the data in a Person.Object directly. This breaks encapsulation and also illustrates that Ada completely separates the concepts of *encapsulation* and *type*. Here is a version of Person.Object that encapsulates its data:

```
package Person is
   type Object is tagged private;
   procedure Put (O : Object);
private
   type Object is tagged
     record
         Name   : String (1 .. 10);
         Gender : Gender_Type;
     end record;
end Person;
```

Because the type Person.Object is tagged, it is possible to create a record extension, which is a derived type with additional data.

```
with Person;
package Programmer is
   type Object is new Person.Object with private;
private
   type Object is new Person.Object with
     record
        Skilled_In : Language_List;
     end record;
end Programmer;
```

The type `Programmer.Object` inherits the data and behavior, i.e. the type's primitive operations, from `Person.Object`; it is thus possible to write:

```
with Programmer;
procedure Main is
   Me : Programmer.Object;
begin
   Programmer.Put (Me);
   Me.Put; --equivalent to the above, Ada 2005 only
end Main;
```

So the declaration of the type `Programmer.Object`, as a record extension of `Person.Object`, implicitly declares a **`procedure`** `Put` that applies to a `Programmer.Object`.

Like in the case of untagged types, objects of type Person and Programmer are convertible. However, where untagged objects are convertible in either direction, conversion of tagged types only works in the direction to the root. (Conversion away from the root would have to add components out of the blue.) Such a conversion is called a *view conversion*, because components are not lost, they only become invisible.

Extension aggregates have to be used if you go away from the root.

22.1.5 Overriding

Now that we have introduced tagged types, record extensions and primitive operations, it becomes possible to understand overriding. In the examples above, we introduced a type `Person.Object` with a primitive operation called `Put`. Here is the body of the package:

```
with Ada.Text_IO;
package body Person is
   procedure Put (O : Object) is
   begin
      Ada.Text_IO.Put (O.Name);
      Ada.Text_IO.Put (" is a ");
      Ada.Text_IO.Put_Line (Gender_Type'Image (O.Gender));
   end Put;
end Person;
```

As you can see, this simple operation prints both data components of the record type to standard output. Now, remember that the record extension `Programmer.Object` has an additional data member. If we write:

```
with Programmer;
procedure Main is
   Me : Programmer.Object;
begin
   Programmer.Put (Me);
   Me.Put; --equivalent to the above, Ada 2005 only
end Main;
```

then the program will call the inherited primitive operation `Put`, which will print the name and gender *but not the additional data*. In order to provide this extra behavior, we must *override* the inherited procedure `Put` like this:

```
with Person;
package Programmer is
   type Object is new Person.Object with private;
   overriding --Optional keyword, new in Ada 2005
   procedure Put (O : Object);
private
   type Object is new Person.Object with
     record
        Skilled_In : Language_List;
     end record;
end Programmer;

package body Programmer is
```

```
   procedure Put (O : Object) is
   begin
      Person.Put (Person.Object (O)); --view conversion to the ancestor type
      Put (O.Skilled_In); --presumably declared in the same package as Language_List
   end Put;
end Programmer;
```

`Programmer.Put` *overrides* `Person.Put`; in other words it *replaces* it completely. Since the intent is to extend the behavior rather than replace it, `Programmer.Put` calls `Person.Put` as part of its behavior. It does this by converting its parameter from the type `Programmer.Object` to its ancestor type `Person.Object`. This construct is a *view conversion*; contrary to a normal type conversion, it does *not* create a new object and does *not* incur any run-time cost. Of course, it is optional that an overriding operation call its ancestor; there are cases where the intent is indeed to replace, not extend, the inherited behavior.

(Note that also for untagged types, overriding of inherited operations is possible. The reason why it's discussed here is that derivation of untagged types is done rather seldom.)

22.1.6 Polymorphism, class-wide programming and dynamic dispatching

The full power of object orientation is realized by polymorphism, class-wide programming and dynamic dispatching, which are different words for the same, single concept. To explain this concept, let us extend the example from the previous sections, where we declared a base tagged type `Person.Object` with a primitive operation `Put` and a record extension `Programmer.Object` with additional data and an overriding primitive operation `Put`.

Now, let us imagine a collection of persons. In the collection, some of the persons are programmers. We want to traverse the collection and call `Put` on each person. When the person under consideration is a programmer, we want to call `Programmer.Put`; when the person is not a programmer, we want to call `Person.Put`. This, in essence, is polymorphism, class-wide programming and dynamic dispatching.

Ada implements this concept by means of *class-wide types*.

Each tagged type, such as `Person.Object`, has a corresponding *class of types* which is the set of types comprising the type `Person.Object` itself and all types that extend `Person.Object`. In our example, this class consists of two types:

- `Person.Object`
- `Programmer.Object`

Ada 95 defines the `Person.Object'Class` attribute to denote the corresponding class-wide type. In other words:

```
declare
   Someone : Person.Object'Class := ...; --to be expanded later
begin
   Someone.Put; --dynamic dispatching
end;
```

The declaration of Someone denotes an object that may be of *either* type, `Person.Object` or `Programmer.Object`. Consequently, the call to the primitive operation `Put` dispatches dynamically to either `Person.Put` or `Programmer.Put`.

The only problem is that, since we don't know whether Someone is a programmer or not, we don't know how many data components Someone has, either, and therefore we don't know how many bytes Someone takes in memory. For this reason, the class-wide type `Person.Object'Class` is *indefinite*[1]. It is impossible to declare an object of this type without giving some constraint. It is, however, possible to:

- declare an object of a class-wide with an initial value (as above). The object is then constrained by its initial value.
- declare an *access value* to such an object (because the access value has a known size);
- pass objects of a class-wide type as parameters to subprograms
- assign an object of a specific type (in particular, the result of a function call) to a variable of a class-wide type.

With this knowledge, we can now build a polymorphic collection of persons; in this example we will quite simply create an array of access values to persons:

```
with Person;
procedure Main is
   type Person_Access is access Person.Object'Class;
   type Array_Of_Persons  is array (Positive range <>) of Person_Access;

   function Read_From_Disk return Array_Of_Persons is separate;

   Everyone : constant Array_Of_Persons := Read_From_Disk;
begin --Main
   for K in Everyone'Range loop
      Everyone (K).all.Put; --dereference followed by dynamic dispatching
   end loop;
end Main;
```

The above procedure achieves our desired goal: it traverses the array of Persons and calls the procedure `Put` that is appropriate for each person.

Advanced topic: How dynamic dispatching works

You don't need to know how dynamic dispatching works in order to use it effectively but, in case you are curious, here is an explanation.

The first component of each object in memory is the *tag*; this is why objects are of a *tagged* type rather than plain records. The tag really is an access value to a table; there is one table for each specific type. The table contains access values to each primitive operation of the type. In our example, since there are two types `Person.Object` and `Programmer.Object`, there are two tables, each containing a single access value. The table for `Person.Object` contains an access value to `Person.Put` and the table for `Programmer.Object` contains an access value to `Programmer.Put`. When you compile your program, the compiler constructs both tables and places them in the program executable code.

1 `http://en.wikibooks.org/wiki/ada%20Programming%2FSubtypes%23Indefinite_subtype`

Each time the program creates a new object of a specific type, it automatically sets its tag to point to the appropriate table.

Each time the program calls a primitive operation, the compiler inserts object code that:

- dereferences the tag to find the table of primitive operations for the specific type of the object at hand
- dereferences the access value to the primitive operation
- calls the primitive operation.

When you perform a view conversion to an ancestor type, the compiler performs these two dereferences at compile time rather than run time: this is *static dispatching*; the compiler emits code that directly calls the primitive operation of the ancestor type specified in the view conversion.

Redispatching

Dispatching works on the (hidden) tag of the object. So what happens when a primitive operation Op1 calls another primitive operation Op2? Which operation will be called when Op1 is called by dispatching?

```
type Root is tagged private;
procedure Op1 (R: Root);
procedure Op2 (R: Root);

type Derived is new Root with private;
-- Derived inherits Op1
overriding procedure Op2 (D: Derived);

procedure Op1 (R: Root) is
begin
  ...
  Op2 (R);               -- not redispatching
  Op2 (Root'Class (R));  -- redispatching
  ...
end Op1;

D: Derived;
C: Root'Class := D;

Op1 (D);  -- static call
Op1 (C);  -- dispatching call
```

In this fragment, Op1 is not overridden, whereas Op2 is overridden. The body of Op1 calls Op2, thus which Op2 will be called for a call of Op1 with a parameter of type Derived?

The answer is: Ada gives complete control over dispatching and redispatching. If you want redispatching, it has to be required explicitly by converting the parameter to the class-wide type again. (Remember: View conversions never lose components, they just hide them. A conversion to the class-wide type makes them visible again.)

Thus the first call of Op1 (statically linked, i.e. not dispatching) calls the inherited Op1 — and within Op1, the first call to Op2 is therefore also a static call to the inherited Op2 (there is no redispatching). However the second call, since the parameter R is converted to the class-wide type, dispatches to the overriding Op2.

The second call of Op1 is a dispatching call to the inherited Op1 and behaves exactly as the first.

To understand what happens here, the implicitly defined inherited Op1 is just the parent operation called with a view conversion of the parameter:

```
procedure Op1 (D: Derived) is
begin
  Op1 (Root (R));  -- view conversion
end Op1;
```

Run-time type identification

Run-time type identification allows the program to (indirectly or directly) query the tag of an object at run time to determine which type the object belongs to. This feature, obviously, makes sense only in the context of polymorphism and dynamic dispatching, so works only on tagged types.

You can determine whether an object belongs to a certain class of types, or to a specific type, by means of the membership test **in**, like this:

```
type Base    is tagged private;
type Derived is new Base    with private;
type Leaf    is new Derived with private;

...
procedure Explicit_Dispatch (This : in Base'Class) is
begin
   if This in Leaf then ... end if;
   if This in Derived'Class then ... end if;
end Explicit_Dispatch;
```

Thanks to the strong typing rules of Ada, run-time type identification is in fact rarely needed; the distinction between class-wide and specific types usually allows the programmer to ensure objects are of the appropriate type without resorting to this feature.

Additionally, the reference manual defines `package Ada.Tags` (RM 3.9(6/2)), attribute `'Tag` (RM 3.9(16,18)), and `function Ada.Tags.Generic_Dispatching_Constructor` (RM 3.9(18.2/2)), which enable direct manipulation with tags.

22.1.7 Creating Objects

The Language Reference Manual's section on 3.3 Objects and Named Numbers ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-3.html`} states when an object is created, and destroyed again. This subsection illustrates how objects are created.

The LRM section starts,

> *Objects are created at run time and contain a value of a given type. An object* *be created and initialized as part of elaborating a declaration, evaluating an allo* *aggregate, or function_call.*

For example, assume a typical hierarchy of object oriented types: a top-level type `Person`, a `Programmer` type derived from `Person`, and possibly more kinds of persons. Each person has a name; assume `Person` objects to have a `Name` component. Likewise, each `Person` has a `Gender` component. The `Programmer` type inherits the components and the operations of the `Person` type, so `Programmer` objects have a `Name` and a `Gender` component, too. `Programmer` objects may have additional components specific to programmers.

Objects of a tagged type are created the same way as objects of any type. The second LRM sentence says, for example, that an object will be created when you declare a variable or a constant of a type. For the tagged type `Person`,

```
declare
   P:  Person;
begin
   Text_IO. Put_Line( "The name is " &  P. Name) ;
end;
```

Nothing special so far. Just like any ordinary variable declaration this O-O one is elaborated. The result of elaboration is an object named `P` of type `Person`. However, `P` has only default name and gender value components. These are likely not useful ones. One way of giving initial values to the object's components is to assign an aggregate.

```
declare
   P:  Person :=  ( Name =>  "Scorsese",  Gender =>  Male) ;
begin
   Text_IO. Put_Line( "The name is " &  P. Name) ;
end;
```

The parenthesized expression after := is called an *aggregate* (4.3 Aggregates ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-3.html`}).

Another way to create an object that is mentioned in the LRM paragraph is to call a function. An object will be created as the return value of a function call. Therefore, instead of using an aggregate of initial values, we might call a function returning an object.

Introducing proper O-O information hiding, we change the package containing the `Person` type so that `Person` becomes a private type. To enable clients of the package to construct `Person` objects we declare a function that returns them. (The function may do some interesting construction work on the objects. For instance, the aggregate above will most probably raise the exception Constraint_Error depending on the name string supplied; the function can mangle the name so that it matches the declaration of the component.) We also declare a function that returns the name of `Person` objects.

```
package Persons is

   type Person is tagged private;

   function Make ( Name:   String;  Sex:   Gender_Type)  return Person;

   function Name ( P:  Person)  return String;

private
```

```
   type Person is tagged
      record
         Name   :   String ( 1 ..   10) ;
         Gender :   Gender_Type;
      end record;

end Persons;
```

Calling the `Make` function results in an object which can be used for initialization. Since the `Person` type is **private** we can no longer refer to the `Name` component of `P`. But there is a corresponding function `Name` declared with type `Person` making it a socalled primitive operation. (The component and the function in this example are both named `Name` However, we can choose a different name for either if we want.)

```
declare
   P:  Person :=  Make ( Name =>  "Orwell",  Sex =>  Male) ;
begin
   Text_IO. Put_Line( "The name is " &  Name( P) ) ;
end;
```

Objects can be copied into another. The target object is first destroyed. Then the component values of the source object are assigned to the corresponding components of the target object. In the following example, the default initialized `P` gets a copy of one of the objects created by the `Make` calls.

```
declare
   P:  Person;
begin
   if 2001 > 1984 then
      P :=  Make ( Name =>  "Kubrick",  Sex =>  Male) ;
   else
      P :=  Make ( Name =>  "Orwell",  Sex =>  Male) ;
   end if;

   Text_IO. Put_Line( "The name is " &  Name( P) ) ;
end;
```

So far there is no mention of the `Programmer` type derived from `Person`. There is no polymorphism yet, and likewise initialization does not yet mention inheritance. Before dealing with `Programmer` objects and their initialization a few words about class-wide types are in order.

22.1.8 More details on primitive operations

Remember what we said before about "Primitive Operations"[2]. Primitive operations are:

- subprograms taking a parameter of the tagged type;
- functions returning an object of the tagged type;
- subprograms taking a parameter of an *anonymous access type* to the tagged type;

2 Chapter 22.1.2 on page 188

- In Ada 2005 only, functions returning an *anonymous access type* to the tagged type;

Additionally, primitive operations must be declared before the type is *frozen* (the concept of freezing will be explained later):

Examples:

```
package X is
   type Object is tagged null record;

   procedure Primitive_1 (This : in     Object);
   procedure Primitive_2 (That :    out Object);
   procedure Primitive_3 (Me   : in out Object);
   procedure Primitive_4 (Them : access Object);
   function  Primitive_5 return Object;
   function  Primitive_6 (Everyone : Boolean) return access Object;
end X;
```

All of these subprograms are primitive operations.

A primitive operation can also take parameters of the same or other types; also, the controlling operand does not have to be the first parameter:

```
package X is
   type Object is tagged null record;

   procedure Primitive_1 (This : in Object; Number : in Integer);
   procedure Primitive_2 (You  : in Boolean; That : out Object);
   procedure Primitive_3 (Me, Her : in out Object);
end X;
```

The definition of primitive operations specifically excludes named access types and class-wide types as well as operations not defined immediately in the same declarative region. Counter-examples:

```
package X is
   type Object is tagged null record;
   type Object_Access is access Object;
   type Object_Class_Access is access Object'Class;

   procedure Not_Primitive_1 (This : in     Object'Class);
   procedure Not_Primitive_2 (This : in out Object_Access);
   procedure Not_Primitive_3 (This :    out Object_Class_Access);
   function  Not_Primitive_4 return Object'Class;

   package Inner is
       procedure Not_Primitive_5 (This : in Object);
   end Inner;
end X;
```

Advanced topic: Freezing rules

Freezing rules (`http://www.adaic.com/standards/05rm/html/RM-13-14.html` ARM 13.14[3]) are perhaps the most complex part of the Ada language definition; this is because the standard tries to describe freezing as unambiguously as possible. Also, that part of the language definition deals with freezing of all entities, including complicated situations like generics and objects reached by dereferencing access values. You can, however, get an intuitive understanding of freezing of tagged types if you understand how dynamic dispatching works[4]. In that section, we saw that the compiler emits a table of primitive operations for each tagged type. The point in the program text where this happens is the point where the tagged type is *frozen*, i.e. the point where the table becomes complete. After the type is frozen, no more primitive operations can be added to it.

This point is the earliest of:

- the end of the package spec where the tagged type is declared
- the appearance of the first type derived from the tagged type

Example:

```
package X is

  type Object is tagged nullrecord;
  procedure Primitive_1 (This: in Object);

  -- this declaration freezes Object
  type Derived is new Object with null record;

  -- illegal: declared after Object is frozen
  procedure Primitive_2 (This: in Object);

end X;
```

Intuitively: at the point where Derived is declared, the compiler starts a new table of primitive operations for the derived type. This new table, initially, is equal to the table of the primitive operations of the parent type, `Object`. Hence, `Object` must freeze.

- the declaration of a variable of the tagged type

Example:

```
package X is

  type Object is tagged null record;
  procedure Primitive_1 (This: in Object);

  V: Object; -- this declaration freezes Object

  -- illegal: declared after Object is frozen
  procedure Primitive_2 (This: in Object);

end X;
```

3 http://en.wikibooks.org/wiki/%20ARM%2013.14

4 Chapter 22.1.6 on page 193

Intuitively: after the declaration of `V`, it is possible to call any of the primitive operations of the type on `V`. Therefore, the list of primitive operations must be known and complete, i.e. frozen.

- The completion (*not* the declaration, if any) of a constant of the tagged type:

```
package X is

  type Object is tagged null record;
  procedure Primitive_1 (This: in Object);

  -- this declaration does NOT freeze Object
  Deferred_Constant: constant Object;

  procedure Primitive_2 (This : in Object); -- OK

private

  -- only the completion freezes Object
  Deferred_Constant: constant Object := (null record);

  -- illegal: declared after Object is frozen
  procedure Primitive_3 (This: in Object);

 end X;
```

22.1.9 New features of Ada 2005

This language feature is only available in Ada 2005

Ada 2005 adds overriding indicators, allows anonymous access types in more places and offers the object.method notation.

Overriding indicators

The new keyword **overriding** can be used to indicate whether an operation overrides an inherited subprogram or not. Its use is optional because of upward-compatibility with Ada 95. For example:

```
package X is
    type Object is tagged null record;

   function  Primitive return access Object; --new in Ada 2005

   type Derived_Object is new Object with null record;

   not overriding --new optional keywords in Ada 2005
   procedure Primitive (This : in Derived_Object); --new primitive operation

   overriding
   function  Primitive return access Derived_Object;
end X;
```

The compiler will check the desired behaviour.

This is a good programming practice because it avoids some nasty bugs like not overriding an inherited subprogram because the programmer spelt the identifier incorrectly, or because a new parameter is added later in the parent type.

It can also be used with abstract operations, with renamings, or when instantiating a generic subprogram:

```
not overriding
procedure Primitive_X (This : in Object) is abstract;

overriding
function  Primitive_Y return Object renames Some_Other_Subprogram;

not overriding
procedure Primitive_Z (This : out Object)
      is new Generic_Procedure (Element => Integer);
```

Object.Method notation

We have already seen this notation:

```
package X is
   type Object is tagged null record;

   procedure Primitive_1 (This: in Object; That: in Boolean);
end X;
```

```
with X;
procedure Main is
   Obj : X.Object;
begin
   Obj.Primitive (That => True); --Ada 2005 object.method notation
end Main;
```

This notation is only available for primitive operations where the controlling parameter is the *first* parameter.

22.1.10 Abstract types

A tagged type can also be abstract (and thus can have abstract operations):

```
package X is

   type Object is abstract tagged ...;

   procedure One_Class_Member      (This : in     Object);
   procedure Another_Class_Member  (This : in out Object);
   function  Abstract_Class_Member return Object  is abstract;

end X;
```

An abstract operation cannot have any body, so derived types are forced to override it (unless those derived types are also abstract). See next section about interfaces for more information about this.

The difference with a non-abstract tagged type is that you cannot declare any variable of this type. However, you can declare an access to it, and use it as a parameter of a class-wide operation.

22.1.11 Multiple Inheritance via Interfaces

This language feature is only available in Ada 2005

Interfaces allow for a limited form of multiple inheritance (taken from Java). On a semantic level they are similar to an "abstract tagged null record" as they may have primitive operations but cannot hold any data and thus these operations cannot have a body, they are either declared **abstract** or **null**. *Abstract* means the operation has to be overridden, *null* means the default implementation is a null body, i.e. one that does nothing.

An interface is declared with:

```
package Printable is

   type Object is interface;

   procedure Class_Member_1 (This : in     Object) is abstract;
   procedure Class_Member_2 (This :    out Object) is null;

end Printable;
```

You implement an **interface** by adding it to a concrete *class*:

```
with Person;

package Programmer is

   type Object is new Person.Object
                  and Printable.Object
   with
      record
         Skilled_In : Language_List;
      end record;

   overriding
   procedure Class_Member_1   (This : in Object);

   not overriding
   procedure New_Class_Member (This : Object; That : String);

end Programmer;
```

As usual, all inherited abstract operations must be overridden although *null subprograms* ones need not.

Such a type may implement a list of interfaces (called the *progenitors*), but can have only one *parent*. The parent may be a concrete type or also an interface.

```
type Derived is new Parent and Progenitor_1 and Progenitor_2 ... with ...;
```

22.1.12 Multiple Inheritance via Mix-in

Ada supports multiple inheritance of *interfaces* (see above), but only single inheritance of *implementation*. This means that a tagged type can *implement* multiple interfaces but can only *extend* a single ancestor tagged type.

This can be problematic if you want to add behavior to a type that already extends another type; for example, suppose you have

```
type Base is tagged private;
type Derived is new Base with private;
```

and you want to make `Derived` controlled, i.e. add the behavior that `Derived` controls its initialization, assignment and finalization. Alas you cannot write:

```
type Derived is new Base and Ada.Finalization.Controlled with private; --illegal
```

since `Ada.Finalization` for historical reasons does not define interfaces `Controlled` and `Limited_Controlled`, but abstract types.

If your base type is not limited, there is no good solution for this; you have to go back to the root of the class and make it controlled. (The reason will become obvious presently.)

For limited types however, another solutions is the use of a mix-in:

```
type Base is tagged limited private;
type Derived;

type Controlled_Mix_In (Enclosing: access Derived) is
  new Ada.Finalization.Limited_Controlled with null record;

overriding procedure Initialize (This: in out Controlled_Mix_In);
overriding procedure Finalize   (This: in out Controlled_Mix_In);

type Derived is new Base with record
  Mix_In: Controlled_Mix_In (Enclosing => Derived'Access); --special syntax here
  --other components here...
end record;
```

This special kind of mix-in is an object with an access discriminant that references its enclosing object (also known as *Rosen trick*). In the declaration of the `Derived` type, we initialize this discriminant with a special syntax: `Derived'Access` really refers to an a access value to the *current instance* of type `Derived`. Thus the access discriminant allows the mix-in to see its enclosing object and all its components; therefore it can initialize and

finalize its enclosing object:

```
overriding procedure Initialize (This: in out Controlled_Mix_In) is
  Enclosing: Derived renames This.Enclosing.all;
begin
  --initialize Enclosing...
end Initialize;
```

and similarly for `Finalize`.

The reason why this does not work for non-limited types is the self-referentiality via the discriminant. Imagine you have two variables of such a non-limited type and assign one to the other:

```
X := Y;
```

In an assignment statement, `Adjust` is called only *after* `Finalize` of the target `X` and so cannot provide the new value of the discriminant. Thus `X.Mixin_In.Enclosing` will inevitably reference `Y`.

Now let's further extend our hierarchy:

```
type Further is new Derived with null record;

overriding procedure Initialize (This: in out Further);
overriding procedure Finalize   (This: in out Further);
```

Oops, this does not work because there are no corresponding procedures for `Derived`, yet - so let's quickly add them.

```
type Base is tagged limited private;
type Derived;

type Controlled_Mix_In (Enclosing: access Derived) is
  new Ada.Finalization.Limited_Controlled with null record;

overriding procedure Initialize (This: in out Controlled_Mix_In);
overriding procedure Finalize   (This: in out Controlled_Mix_In);

type Derived is new Base with record
  Mix_In: Controlled_Mix_In (Enclosing => Derived'Access);  --special syntax here
  --other components here...
end record;

not overriding procedure Initialize (This: in out Derived);  --sic, they are new
not overriding procedure Finalize   (This: in out Derived);

type Further is new Derived with null record;

overriding procedure Initialize (This: in out Further);
overriding procedure Finalize   (This: in out Further);
```

We have of course to write `not overriding` for the procedures on `Derived` because there is indeed nothing they could override. The bodies are

```
not overriding procedure Initialize (This: in out Derived) is
begin
  --initialize Derived...
end Initialize;

overriding procedure Initialize (This: in out Controlled_Mix_In) is
  Enclosing: Derived renames This.Enclosing.all;
begin
  Initialize (Enclosing);
end Initialize;
```

To our dismay, we have to learn that `Initialize/Finalize` for objects of type `Further` will not be called, instead those for the parent `Derived`. Why?

```
declare
  X: Further;  -- Initialize (Derived (X)) is called here
begin
  null;
end;  -- Finalize (Derived (X)) is called here
```

The reason is that the mix-in defines the local object `Enclosing` to be of type `Derived` in the renames-statement above. To cure this, we have necessarily to use the dreaded redispatch (shown in different but equivalent notations):

```
overriding procedure Initialize (This: in out Controlled_Mix_In) is
  Enclosing: Derived renames This.Enclosing.all;
begin
  Initialize (Derived'Class (Enclosing));
end Initialize;

overriding procedure Finalize (This: in out Controlled_Mix_In) is
  Enclosing: Derived'Class renames Derived'Class (This.Enclosing.all);
begin
  Enclosing.Finalize;
end Finalize;

declare
  X: Further;  -- Initialize (X) is called here
begin
  null;
end;  -- Finalize (X) is called here
```

Alternatively (and presumably better still) is to write

```
type Controlled_Mix_In (Enclosing: access Derived'Class) is
  new Ada.Finalization.Limited_Controlled with null record;
```

Then we automatically get redispatch and can omit the type conversions on `Enclosing`.

22.2 Class names

Class names

Both the class package and the class record need a name. In theory they may have the same name, but in practice this leads to nasty (because of unintutive error messages) name clashes when you use the **use** clause. So over time three de facto naming standards have been commonly used.

22.2.1 Classes/Class

The package is named by a plural noun and the record is named by the corresponding singular form.

```
package Persons is

   type Person is tagged
      record
         Name   : String (1 .. 10);
         Gender : Gender_Type;
      end record;

end Persons;
```

This convention is the usually used in Ada's built-in libraries.

Disadvantage: Some "multiples" are tricky to spell, especially for those of us who aren't native English speakers.

22.2.2 Class/Object

The package is named after the class, the record is just named Object.

```
package Person is

   type Object is tagged
      record
         Name   : String (1 .. 10);
         Gender : Gender_Type;
      end record;

end Person;
```

Most UML[5] and IDL[6] code generators use this technique.

Disadvantage: You can't use the **use** clause on more than one such class packages at any one time. However you can always use the "type" instead of the package.

5 http://en.wikipedia.org/wiki/Unified%20Modeling%20Language
6 http://en.wikipedia.org/wiki/Interface%20description%20language

22.2.3 Class/Class_Type

The package is named after the class, the record is postfixed with *_Type*.

```
package Person is

   type Person_Type is tagged
      record
         Name   : String (1 .. 10);
         Gender : Gender_Type;
      end record;

end Person;
```

Disadvantage: lots of ugly "_Type" postfixes.

22.3 Object-Oriented Ada for C++ programmers

Object-Oriented Ada for C++ programmers

In C++, the construct

```
class C {
  virtual void v();
  void w();
  static void u();
};
```

is strictly equivalent to the following in Ada:

```
package P is
  type C is tagged null record;
  procedure V (This : C);        --primitive operation, will be inherited upon derivation
  procedure W (This : C'Class);  --not primitive, will not be inherited upon derivation
  procedure U;
end P;
```

In C++, member functions implicitly take a parameter `this` which is of type `C*`. In Ada, all parameters are explicit. As a consequence, the fact that `u()` does *not* take a parameter is implicit in C++ but explicit in Ada.

In C++, `this` is a pointer. In Ada, the explicit `This` parameter does not have to be a pointer; all parameters of a tagged type are implicitly passed by reference anyway.

22.3.1 Static dispatching

In C++, function calls dispatch statically in the following cases:

- the target of the call is an object type
- the member function is non-virtual

For example:

```
C object;
object.v();
object.w();
```

both dispatch statically. In particular, the static dispatch for v() may be confusing; this is because object is neither a pointer nor a reference. Ada behaves exactly the same in this respect, except that Ada calls this *static binding* rather than *dispatching*:

```
declare
   Object : P.C;
begin
   Object.V; --statically bound
   Object.W; --statically bound
end;
```

22.3.2 Dynamic dispatching

In C++, a function call dispatches dynamically if the two following conditions are met simultaneously:

- the target of the call is a pointer or a reference
- the member function is virtual.

For example:

```
C* object;
object->v(); // dynamic dispatch
object->w(); // static, non-virtual member function
object->u(); // illegal: static member function
C::u(); // static dispatch
```

In Ada, a primitive subprogram call dispatches (dynamically) if and only if:

- the target object is of a class-wide type;

Note: In Ada vernacular, the term *dispatching* always means *dynamic*.

For example:

```
declare
   Object : P.C'Class := ...;
begin
   P.V (Object); --dispatching
   P.W (Object); --statically bound:  not a primitive operation
   P.U; --statically bound
end;
```

As can be seen *there is no need for access types or pointers* to do dispatching in Ada. In Ada, *tagged types are always passed by-reference to subprograms* without the need for explicit access values.

Also note that in C++, the class serves as:

- the unit of encapsulation (Ada uses packages and visibility for this)
- the type, like in Ada.

As a consequence, you call C::u() in C++ because u() is encapsulated in C, but P.U in Ada since U is encapsulated in the *package* P, not the *type* C.

22.3.3 Class-wide and specific types

The most confusing part for C++ programmers is the concept of a "class-wide type". To help you understand:

- pointers and references in C++ are really, implicitly, class-wide;
- object types in C++ are really specific;
- C++ provides no way to declare the equivalent of:

```
type C_Specific_Access is access C;
```

- C++ provides no way to declare the equivalent of:

```
type C_Specific_Access_One is access C;
type C_Specific_Access_Two is access C;
```

which, in Ada, are two different, *incompatible* types, possibly allocating their memory from different storage pools!

- In Ada, you do *not* need access values for dynamic dispatching.
- In Ada, you use access values for dynamic memory management (only) and class-wide types for dynamic dispatching (only).
- In C++, you use pointers and references both for dynamic memory management and for dynamic dispatching.
- In Ada, class-wide types are explicit (with `'Class`).
- In C++, class-wide types are implicit (with `*` or `&`).

22.3.4 Constructors

in C++, a special syntax declares a constructor:

```
class C {
   C(/* optional parameters */); // constructor
};
```

A constructor cannot be virtual. A class can have as many constructors, differentiated by their parameters, as necessary.

Ada does not have such constructors. Perhaps they were not deemed necessary since in Ada, any function that returns an object of the tagged type can serve as a kind of constructor. This is however not the same as a real constructor like the C++ one; this difference is most striking in cases of derivation trees (see Finalization below). The Ada constructor

subprograms do not have to have a special name and there can be as many constructors as necessary; each function can take parameters as appropriate.

```
package P is
  type T is tagged private;
  function Make                 return T;  -- constructor
  function To_T (From: Integer) return T;  -- another constructor
  -- procedure Make (This: out T);         -- not a constructor
private
  ...
end P;
```

If an Ada constructor function is also a primitive operation (as in the example above), it becomes abstract upon derivation and has to be overridden if the derived type is not itself abstract. If you do not want this, declare such functions in a nested scope.

In C++, one idiom is the *copy constructor* and its cousin the *assignment operator*:

```
class C {
   C(const C& that); // copies "that" into "this"
   C& operator= (const C& right); // assigns "right" to "this",
which is "left"
};
```

This copy constructor is invoked implicitly on initialization, e.g.

```
C a = b; // calls the copy constructor
C c;
a = c;   // calls the assignment operator
```

Ada provides a similar functionality by means of *controlled types*. A controlled type is one that extends the predefined type `Ada.Finalization.Controlled`:

```
with Ada.Finalization;
package P is
  type T is new Ada.Finalization.Controlled with private;
  function Make return T;  -- constructor
private
  type T is ... end record;
  overriding procedure Initialize (This: in out T);
  overriding procedure Adjust     (This: in out T); -- copy contructor
end P;
```

Note that Initialize is not a constructor; it resembles the C++ constructor in some way, but is also very different. Suppose you have a type T1 derived from T with an appropriate overriding of Initialize. A real constructor (like the C++ one) would automatically first construct the parent components (T), then the child components. In Ada, this is not automatic. In order to mimic this in Ada, we have to write:

```
procedure Initialize (This: in out T1) is
begin
```

```
   Initialize (T (This));  -- Don't forget this part!
   ...  -- handle the new components here
end Initialize;
```

The compiler inserts a call to Initialize after each object of type T is allocated when no initial value is given. It also inserts a call to Adjust after each assignment to the object. Thus, the declarations:

```
A: T;
B: T := X;
```

will:

- allocate memory for A
- call Initialize (A)
- allocate memory for B
- copy the contents of X to B
- call Adjust (B)

Initialize (B) will not be called because of the explicit initialization.

So, the equivalent of a copy constructor is an overriding of Adjust.

If you would like to provide this functionality to a type that extends another, non-controlled type, see "Multiple Inheritance"[7].

22.3.5 Destructors

In C++, a destructor is a member function with only the implicit `this` parameter:

```
class C {
   virtual ~C(); // destructor
}
```

While a constructor *cannot* be virtual, a destructor *must* be virtual. Unfortunately, the rules of the C++ language do not enforce this, so it is quite easy for a programmer to wreak havoc in their programs by simply forgetting the keyword `virtual`.

In Ada, the equivalent functionality is again provided by controlled types, by overriding the procedure Finalize:

```
with Ada.Finalization;
package P is
   type T is new Ada.Finalization.Controlled with private;
   function Make return T;  -- constructor
private
   type T is ... end record;
```

7 Chapter 22.4.2 on page 217

```
   overriding procedure Finalize (This: in out T);  -- destructor
end P;
```

Because Finalize is a primitive operation, it is automatically "virtual"; you cannot, in Ada, forget to make a destructor virtual.

22.3.6 Encapsulation: public, private and protected members

In C++, the unit of encapsulation is the class; in Ada, the unit of encapsulation is the package. This has consequences on how an Ada programmer places the various components of an object type.

```
class C {
public:
   int a;
   void public_proc();
protected:
   int b;
   int protected_func();
private:
   bool c;
   void private_proc();
};
```

A way to mimic this C++ class in Ada is to define a hierarchy of types, where the base type is the public part, which must be abstract so that no stand-alone objects of this base type can be defined. It looks like so:

```
private with Ada.Finalization;

package CPP is

  type Public_Part is abstract tagged record  -- no objects of this type
    A: Integer;
  end record;

  procedure Public_Proc (This: in out Public_Part);

  type Complete_Type is new Public_Part with private;

  -- procedure Public_Proc (This: in out Complete_Type);  --
inherited, implicitly defined

private  -- visible for children

  type Private_Part;  -- declaration stub
  type Private_Part_Pointer is access Private_Part;

  type Private_Component is new Ada.Finalization.Controlled with record
    P: Private_Part_Pointer;
  end record;

  overriding procedure Initialize (X: in out Private_Component);
  overriding procedure Adjust     (X: in out Private_Component);
  overriding procedure Finalize   (X: in out Private_Component);
```

```
  type Complete_Type is new Public_Part with record
    B: Integer;
    P: Private_Component;  -- must be controlled to avoid storage
leaks
  end record;

  not overriding procedure Protected_Proc (This: Complete_Type);

end CPP;
```

The private part is defined as a stub only, its completion is hidden in the body. In order to make it a component of the complete type, we have to use a pointer since the size of the component is still unknown (the size of a pointer is known to the compiler). With pointers, unfortunately, we incur the danger of memory leaks, so we have to make the private component controlled.

For a little test, this is the body, where the subprogram bodies are provided with identifying prints:

```
with Ada.Unchecked_Deallocation;
with Ada.Text_IO;

package body CPP is

  procedure Public_Proc (This: in out Public_Part) is  -- primitive
  begin
    Ada.Text_IO.Put_Line ("Public_Proc" & Integer'Image (This.A));
  end Public_Proc;

  type Private_Part is record  -- complete declaration
    C: Boolean;
  end record;

  overriding procedure Initialize (X: in out Private_Component) is
  begin
    X.P := new Private_Part'(C => True);
    Ada.Text_IO.Put_Line ("Initialize " & Boolean'Image (X.P.C));
  end Initialize;

  overriding procedure Adjust (X: in out Private_Component) is
  begin
    Ada.Text_IO.Put_Line ("Adjust " & Boolean'Image (X.P.C));
    X.P := new Private_Part'(C => X.P.C);  -- deep copy
  end Adjust;

  overriding procedure Finalize (X: in out Private_Component) is
    procedure Free is new Ada.Unchecked_Deallocation (Private_Part,
Private_Part_Pointer);
  begin
    Ada.Text_IO.Put_Line ("Finalize " & Boolean'Image (X.P.C));
    Free (X.P);
  end Finalize;

  procedure Private_Proc (This: in out Complete_Type) is  -- not primitive
  begin
    Ada.Text_IO.Put_Line ("Private_Proc" & Integer'Image (This.A) &
Integer'Image (This.B) & ' ' & Boolean'Image (This.P.P.C));
  end Private_Proc;

  not overriding procedure Protected_Proc (This: Complete_Type) is  -- primitive
    X: Complete_Type := This;
```

```
  begin
    Ada.Text_IO.Put_Line ("Protected_Proc" & Integer'Image (This.A)
& Integer'Image (This.B));
    Private_Proc (X);
  end Protected_Proc;

end CPP;
```

We see that, due to the construction, the private procedure is not a primitive operation.

Let's define a child class so that the protected operation can be reached:

```
package CPP.Child is

  procedure Do_It (X: Complete_Type);  -- not primitive

end CPP.Child;
```

A child can look inside the private part of the parent and thus can see the protected procedure:

```
with Ada.Text_IO;

package body CPP.Child is

  procedure Do_It (X: Complete_Type) is
  begin
    Ada.Text_IO.Put_Line ("Do_It" & Integer'Image (X.A) &
Integer'Image (X.B));
    Protected_Proc (X);
  end Do_It;

end CPP.Child;
```

This is a simple test program, its output is shown below.

```
with CPP.Child;
use  CPP.Child, CPP;

procedure Test_CPP is

  X, Y: Complete_Type;

begin

  X.A := +1;
  Y.A := -1;

  Public_Proc (X);  Do_It (X);
  Public_Proc (Y);  Do_It (Y);

  X := Y;

  Public_Proc (X);  Do_It (X);

end Test_CPP;
```

This is the commented output of the test program:

```
Initialize TRUE                    Test_CPP: Initialize X
Initialize TRUE                                  and Y
Public_Proc 1                      |  Public_Proc (X):  A=1
Do_It 1-1073746208                 |  Do_It (X):        B
uninitialized
Adjust TRUE                        |  |  Protected_Proc (X): Adjust
local copy X of This
Protected_Proc 1-1073746208        |  |  |
Private_Proc 1-1073746208 TRUE     |  |  |  Private_Proc on local
copy of This
Finalize TRUE                      |  |  Protected_Proc (X):
Finalize local copy X
Public_Proc-1                      |  ditto for Y
Do_It-1 65536                      |  |
Adjust TRUE                        |  |
Protected_Proc-1 65536             |  |
Private_Proc-1 65536 TRUE          |  |
Finalize TRUE                      |  |
Finalize TRUE                      |  Assignment: Finalize target
X.P.C
Adjust TRUE                        |  |           Adjust: deep copy
Public_Proc-1                      |  again for X, i.e. copy of Y
Do_It-1 65536                      |  |
Adjust TRUE                        |  |
Protected_Proc-1 65536             |  |
Private_Proc-1 65536 TRUE          |  |
Finalize TRUE                      |  |
Finalize TRUE                      Finalize Y
Finalize TRUE                           and X
```

You see that a direct translation of the C++ behaviour into Ada is difficult, if feasible at all. Methinks, the primitive Ada subprograms corresponds more to virtual C++ methods (in the example, they are not). Each language has its own idiosyncrasies which have to be taken into account, so that attempts to directly translate code from one into the other may not be the best approach.

22.3.7 De-encapsulation: friends and stream input-output

In C++, a friend function or class can see all members of the class it is a friend of. Friends break encapsulation and are therefore to be discouraged. In Ada, since packages and not classes are the unit of encapsulation, a "friend" subprogram is simply one that is declared in the same package as the tagged type.

In C++, stream input and output are the particular case where friends are usually necessary:

```
#include <iostream>
class C {
public:
   C();
   friend ostream& operator<<(ostream& output, C& arg);
private:
   int a, b;
   bool c;
};
```

```
#include <iostream>
int main() {
   C object;
   cout << object;
   return 0;
};
```

Ada does not need this construct because it defines stream input and output operations by default:

```
package P is
   pragma Elaborate_Body; -- explained below
   type C is tagged private;
private
   type C is tagged record
      A, B : Integer;
      C : Boolean;
   end record;
end P;
```

```
with Ada.Text_IO.Text_Streams;
with P;
procedure Main is
   Object : P.C;
begin
   P.C'Output (Stream => Ada.Text_IO.Text_Streams.Stream
(Ada.Text_IO.Default_Output),
               Item => Object);
end Main;
```

By default, the `Output` attribute sends the tag of the object to the stream then calls the more basic `Write` attribute, which sends the components to the stream in the same order as the declaration, i.e. A, B then C. It is possible to override the default implementation of the `Input`, `Output`, `Read` and `Write` attributes like this:

```
with Ada.Streams;
package body P is
   procedure My_Write (Stream : not null access
Ada.Streams.Root_Stream_Type'Class;
                       Item : in C) is
   begin
      -- The default is to write A then B then C; here we change the
ordering.
      Boolean'Write (Stream, Item.C);
      Integer'Write (Stream, Item.B);
      Integer'Write (Stream, Item.A);
   end My_Write;

   for C'Write use My_Write; -- override the default attribute
end P;
```

In the above example, `P.C'output` calls `P.C'Write` which is overridden in the body of the package. Since the specification of package `P` does not define any subprograms, it does not normally need a body, so a package body is forbidden. The `pragma Elaborate_Body` tells the compiler that this package does have a body that is needed for other reasons.

Note that the stream IO attributes are not primitive operations of the tagged type; this is also the case in C++ where the friend operators are not, in fact, member functions of the type.

22.3.8 Terminology

Ada	**C++**
Package	class (as a unit of encapsulation)
Tagged type	class (of objects) (as a type) (*not* pointer or reference, which are class-wide)
Primitive operation	virtual member function
Tag	pointer to the virtual table
Class (of types)	-
Class-wide type	-
Class-wide operation	static member function
Access value to a specific tagged type	-
Access value to a class-wide type	Pointer or reference to a class

22.4 See also

See also

22.4.1 Wikibook

- Ada Programming[8]
- Ada Programming/Types/record[9]
- record[10]
- interface[11]
- tagged[12]

22.4.2 Wikipedia

- Object-oriented programming[13]

8 http://en.wikibooks.org/wiki/Ada%20Programming
9 Chapter 12 on page 91
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Frecord
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Finterface
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Ftagged
13 http://en.wikipedia.org/wiki/Object-oriented%20programming

22.4.3 Ada Reference Manual

Ada 95

- 3.8 Record Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-8.html}
- 3.9 Tagged Types and Type Extensions ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-9.html}
- 3.9.1 Type Extensions ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-9-1.html}
- 3.9.2 Dispatching Operations of Tagged Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-9-2.html}
- 3.9.3 Abstract Types and Subprograms ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-9-3.html}
- 3.10 Access Types ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-3-10.html}

Ada 2005

- 3.8 Record Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-8.html}
- 3.9 Tagged Types and Type Extensions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-9.html}
- 3.9.1 Type Extensions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-9-1.html}
- 3.9.2 Dispatching Operations of Tagged Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-9-2.html}
- 3.9.3 Abstract Types and Subprograms ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-9-3.html}
- 3.9.4 Interface Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-9-4.html}
- 3.10 Access Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-10.html}

22.4.4 Ada Quality and Style Guide

- Chapter 9: Object-Oriented Features ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_9/}

es:Programación en Ada/Tipos etiquetados[14]

14 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FTipos%20etiquetados

23 New in Ada 2005

This is an overview of the major features that are available in **Ada 2005**, the version of the Ada standard that was accepted by ISO in January 2007 (to differentiate it from its predecessors Ada 83[1] and Ada 95[2], the informal name Ada 2005 is generally agreed on). For the rationale and a more detailed (and very technical) description, see the Amendment[3] to the Ada Reference Manual following the links to the last version of every Ada Issue document (AI).

Although the standard is now published, not all compilers will be able to handle it. Many of these additions are already implemented by the following Free Software[4] compilers:

- GNAT GPL Edition[5]
- GCC 4.1[6]
- GNAT Pro 6.0.2[7] (the AdaCore supported version) is a complete implementation.

After downloading and installing any of them, remember to use the `-gnat05` switch when compiling Ada 2005 code. Note that Ada 2005 is the default mode in GNAT GPL 2007 Edition.

23.1 Language features

Language features

23.1.1 Character set

Not only does Ada 2005 now support a new 32-bit character type — called `Wide_Wide_Character` — but the source code itself may be of this extended character set as well. Thus Russians and Indians, for example, will be able to use their native language in identifiers and comments. And mathematicians will rejoice: The whole Greek and fractur character sets are available for identifiers. For example, Ada.Numerics[8] will be extended with a new constant:

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%2083
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%2095
3 http://www.ada-auth.org/AI-XREF.HTML#Amend_Doc
4 http://en.wikipedia.org/wiki/Free%20Software
5 http://libre.adacore.com/
6 http://gcc.gnu.org/
7 http://www.adacore.com/home/gnatpro/
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics

```
π : constant := Pi;
```

This is not a new idea — GNAT[9] always had the `-gnatic` compiler option to specify the character set `http://gcc.gnu.org/onlinedocs/gnat_ugn_unw/Character-Set-Control.html`. But now this idea has become standard, so all Ada compilers will need to support Unicode 4.0[10] for identifiers — as the new standard requires.

See also:

- AI95-00285-01 Support for 16-bit and 32-bit characters ^{`http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00285.TXT`}
- AI95-00388-01 Add Greek pi to Ada.Numerics ^{`http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00388.TXT`}

23.1.2 Interfaces

Interfaces allow for a limited form of multiple inheritance similar to Java and C#.

You find a full description here: Ada Programming/OO[11].

See also:

- AI95-00251-01 Abstract Interfaces to provide multiple inheritance ^{`http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00251.TXT`}
- AI95-00345-01 Protected and task interfaces ^{`http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00345.TXT`}

23.1.3 Union

In addition to Ada's safe variant record an unchecked C style union is now available.

You can find a full description here: Ada Programming/Types/record#Union[12].

See also:

- AI95-00216-01 Unchecked unions -- variant records with no run-time discriminant ^{`http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00216.TXT`}
- Annex B.3.3 Pragma Unchecked_Union ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-3-3.html`}

9 `http://en.wikipedia.org/wiki/GNAT`
10 `http://en.wikipedia.org/wiki/Unicode`
11 Chapter 22.4.2 on page 217
12 Chapter 12.6 on page 95

23.1.4 With

The with statement got a massive upgrade. First there is the new limited with[13] which allows two packages to *with* each other. Then there is private with[14] to make a package only visible inside the private part of the specification.

See also:

- AI95-00217-01 Limited With Clauses ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00217.TXT}
- AI95-00262-01 Access to private units in the private part ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00262.TXT}

23.1.5 Access types

Not null access

An access type definition can specify that the access type can never be null.

See Ada Programming/Types/access#Not null access[15].

See also: AI95-00231-01 Access-to-constant parameters and null-excluding access subtypes ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00231.TXT}

Anonymous access

The possible uses of anonymous access types are extended. They are allowed virtually in every type or object definition, including access to subprogram parameters. Anonymous access types may point to constant objects as well. Also, they could be declared to be not null.

With the addition of the following operations in package Standard , it is possible to test the equality of anonymous access types.

```
function "=" (Left, Right : universal_access) return Boolean;
function "/="(Left, Right : universal_access) return Boolean;
```

See Ada Programming/Types/access#Anonymous access[16].

See also:

- AI95-00230-01 Generalized use of anonymous access types ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00230.TXT}
- AI95-00385-01 Stand-alone objects of anonymous access types ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00385.TXT}

13 Chapter 17.3.3 on page 138
14 Chapter 17.3.2 on page 138
15 Chapter 13.10.1 on page 110
16 Chapter 13.4 on page 103

- AI95-00318-01 Limited and anonymous access return types ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00318.TXT}

23.2 Language library

Language library

23.2.1 Containers

A major addition to the language library is the generic packages for containers. If you are familiar with the C++ STL, you will likely feel very much at home using Ada . One thing, though: Ada is a block structured language. Many ideas of how to use the STL employ this feature of the language. For example, local subprograms can be supplied to iteration schemes.

The original Ada Issue text AI95-00302-01 Container library ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00302.TXT} has now been transformed into A.18 Containers ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-18.html} .

If you know how to write Ada programs, and have a need for vectors, lists, sets, or maps (tables), please have a look at the AI95-00302-01 AI Text ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00302.TXT} mentioned above. There is an *!example* section in the text explaining the use of the containers in some detail. Matthew Heaney provides a number of demonstration programs with his reference implementation of AI-302 (Ada) which you can find at tigris[17].

In Ada Programming/Containers[18] you will find a demo using containers.

Historical side note: The C++ STL draws upon the work of David R. Musser and Alexander A. Stepanov. For some of their studies of generic programming, they had been using Ada 83. The Stepanov Papers Collection[19] has a few publications available.

23.2.2 Scan Filesystem Directories and Environment Variables

See also:

- AI95-00248-01 Directory Operations ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00248.TXT}
- AI95-00370-01 Environment variables ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00370.TXT}

17 http://charles.tigris.org
18 Chapter 24 on page 231
19 http://www.stepanovpapers.com/

23.2.3 Numerics

Besides the new constant of package Ada.Numerics (see Character Set[20] above), the most important addition are the packages to operate with vectors and matrices.

See also:

- AI95-00388-01 Add Greek pi (π) to Ada.Numerics ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00388.TXT}
- AI95-00296-01 Vector and matrix operations ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00296.TXT}

(Related note on Ada programming tools: AI-388 contains an interesting assessment of how compiler writers are bound to perpetuate the lack of handling of international characters in programming support tools for now. As an author of Ada programs, be aware that your tools provider or Ada consultant could recommend that the program text be 7bit ASCII only.)

23.3 Real-Time and High Integrity Systems

Real-Time and High Integrity Systems

See also:

- AI95-00297-01 Timing events ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00297.TXT}
- AI95-00307-01 Execution-Time Clocks ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00307.TXT}
- AI95-00354-01 Group execution-time budgets ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00354.TXT}
- AI95-00266-01 Task termination procedure ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00266.TXT}
- AI95-00386-01 Further functions returning Time_Span values ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00386.TXT}

23.3.1 Ravenscar profile

See also:

- AI95-00249-01 Ravenscar profile for high-integrity systems ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00249.TXT}
- AI95-00305-01 New pragma and additional restriction identifiers for real-time systems ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00305.TXT}
- AI95-00347-01 Title of Annex H ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00347.TXT}

20 Chapter 23.1.1 on page 219

- AI95-00265-01 Partition Elaboration Policy for High-Integrity Systems ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00265.TXT}

23.3.2 New scheduling policies

See also:

- AI95-00355-01 Priority Specific Dispatching including Round Robin ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00355.TXT}
- AI95-00357-01 Support for Deadlines and Earliest Deadline First Scheduling ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00357.TXT}
- AI95-00298-01 Non-Preemptive Dispatching ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00298.TXT}

23.3.3 Dynamic priorities for protected objects

See also: AI95-00327-01 Dynamic ceiling priorities ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00327.TXT}

23.4 Summary of what's new

Summary of what's new

23.4.1 New keywords

Added 3 keywords (72 total)

- **interface**
- **overriding**
- **synchronized**

23.4.2 New pragmas

Added 11 pragmas:

- **pragma**
- **pragma**
- **pragma**
- **pragma**
- **pragma**
- **pragma**
- **pragma**
- **pragma**
- **pragma**
- **pragma**
- **pragma**

23.4.3 New attributes

Added 7 attributes:

- *Machine_Rounding*
- *Mod*
- *Priority*
- *Stream_Size*
- *Wide_Wide_Image*
- *Wide_Wide_Value*
- *Wide_Wide_Width*

23.4.4 New packages

- Assertions:
  - Ada.Assertions
- Container library:
  - Ada.Containers
  - Ada.Containers.Vectors
  - Ada.Containers.Doubly_Linked_Lists
  - Ada.Containers.Generic_Array_Sort (generic procedure)
  - Ada.Containers.Generic_Constrained_Array_Sort (generic procedure)
  - Ada.Containers.Hashed_Maps
  - Ada.Containers.Ordered_Maps
  - Ada.Containers.Hashed_Sets
  - Ada.Containers.Ordered_Sets
  - Ada.Containers.Indefinite_Vectors
  - Ada.Containers.Indefinite_Doubly_Linked_Lists
  - Ada.Containers.Indefinite_Hashed_Maps
  - Ada.Containers.Indefinite_Ordered_Maps
  - Ada.Containers.Indefinite_Hashed_Sets
  - Ada.Containers.Indefinite_Ordered_Sets
- Vector and matrix manipulation:
  - Ada.Numerics.Real_Arrays
  - Ada.Numerics.Complex_Arrays
  - Ada.Numerics.Generic_Real_Arrays
  - Ada.Numerics.Generic_Complex_Arrays
- General OS facilities:
  - Ada.Directories
  - Ada.Directories.Information
  - Ada.Environment_Variables
- String hashing:
  - Ada.Strings.Hash (generic function)
  - Ada.Strings.Fixed.Hash (generic function)
  - Ada.Strings.Bounded.Hash (generic function)

  - Ada.Strings.Unbounded.Hash (generic function)
  - Ada.Strings.Wide_Hash (generic function)
  - Ada.Strings.Wide_Fixed.Wide_Hash (generic function)
  - Ada.Strings.Wide_Bounded.Wide_Hash (generic function)
  - Ada.Strings.Wide_Unbounded.Wide_Hash (generic function)
  - Ada.Strings.Wide_Wide_Hash (generic function)
  - Ada.Strings.Wide_Wide_Fixed.Wide_Wide_Hash (generic function)
  - Ada.Strings.Wide_Wide_Bounded.Wide_Wide_Hash (generic function)
  - Ada.Strings.Wide_Wide_Unbounded.Wide_Wide_Hash (generic function)
- Time operations:
  - Ada.Calendar.Time_Zones
  - Ada.Calendar.Arithmetic
  - Ada.Calendar.Formatting
- Tagged types:
  - Ada.Tags.Generic_Dispatching_Constructor (generic function)
- Text packages:
  - Ada.Complex_Text_IO
  - Ada.Text_IO.Bounded_IO
  - Ada.Text_IO.Unbounded_IO
  - Ada.Wide_Text_IO.Bounded_IO
  - Ada.Wide_Text_IO.Unbounded_IO
  - Ada.Wide_Characters
  - Ada.Wide_Wide_Characters
- `Wide_Wide_Character` packages:
  - Ada.Strings.Wide_Wide_Bounded
  - Ada.Strings.Wide_Wide_Fixed
  - Ada.Strings.Wide_Wide_Maps
  - Ada.Strings.Wide_Wide_Maps.Wide_Wide_Constants
  - Ada.Strings.Wide_Wide_Unbounded
  - Ada.Wide_Wide_Text_IO
  - Ada.Wide_Wide_Text_IO.Complex_IO
  - Ada.Wide_Wide_Text_IO.Editing
  - Ada.Wide_Wide_Text_IO.Text_Streams
  - Ada.Wide_Wide_Text_IO.Unbounded_IO
- Execution-time clocks:
  - Ada.Execution_Time
  - Ada.Execution_Time.Timers
  - Ada.Execution_Time.Group_Budgets
- Dispatching:
  - Ada.Dispatching
  - Ada.Dispatching.EDF
  - Ada.Dispatching.Round_Robin
- Timing events:
  - Ada.Real_Time.Timing_Events

- Task termination procedures:
  - Ada.Task_Termination

23.5 See also

See also

23.5.1 Wikibook

- Ada Programming/Ada 83[21]
- Ada Programming/Ada 95[22]
- Ada Programming/Ada 2012[23]
- Ada Programming/Object Orientation[24]
- Ada Programming/Types/access[25]
- Ada Programming/Keywords[26]
- Ada Programming/Keywords/and[27]
- Ada Programming/Keywords/interface[28]
- Ada Programming/Attributes[29]
- Ada Programming/Pragmas[30]
- Ada Programming/Pragmas/Restrictions[31]
- Ada Programming/Libraries/Ada.Containers[32]
- Ada Programming/Libraries/Ada.Directories[33]

23.5.2 Pages in the category Ada 2005

- Category:Ada Programming/Ada 2005 feature[34]

23.6 External links

21 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%2083
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%2095
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%202012
24 Chapter 22 on page 187
25 Chapter 13 on page 99
26 Chapter 35 on page 293
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fand
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Finterface
29 Chapter 38 on page 305
30 Chapter 39 on page 317
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FRestrictions
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Directories
34 http://en.wikibooks.org/wiki/%3ACategory%3AAda%20Programming%2FAda%202005%20feature

External links

23.6.1 Papers and presentations

- Ada 2005: Putting it all together[35] (SIGAda 2004 presentation)
- GNAT and Ada 2005[36] (SIGAda 2004 paper)
- An invitation to Ada 2005[37], and the presentation of this paper[38] at Ada-Europe 2004

23.6.2 Rationale

- *Rationale for Ada 2005*[39] by John Barnes[40]:
  1. Introduction
  2. Object Oriented Model
  3. Access Types
  4. Structure and Visibility
  5. Tasking and Real-Time
  6. Exceptions, Generics, Etc.
  7. Predefined Library
  8. Containers
  9. Epilogue

  References

  Index

Available as a single document for printing[41].

23.6.3 Language Requirements

- *Instructions to the Ada Rapporteur Group from SC22/WG9 for Preparation of the Amendment to ISO/IEC 8652*[42] (10 October 2002), and a presentation of this document[43] at SIGAda 2002

23.6.4 Ada Reference Manual

- **Ada Reference Manual**, ISO/IEC 8652:1995(E) with COR.1:2001 and AMD.1:2007[44]

35 http://www.sigada.org/conf/sigada2004/SIGAda2004-CDROM/SIGAda2004-Proceedings/Ada2005Panel.pdf
36 http://www.adacore.com/wp-content/files/attachments/Ada_2005_and_GNAT.pdf
37 http://sigada.org/ada_letters/sept2003/Invitation_to_Ada_2005.pdf
38 http://www.cs.kuleuven.ac.be/~dirk/ada-belgium/events/04/040616-aec-ada2005.pdf
39 http://www.adaic.com/standards/05rat/html/Rat-TTL.html
40 http://en.wikipedia.org/wiki/John%20Barnes%20%28computer%20scientist%29
41 http://www.adaic.com/standards/05rat/Rationale05.pdf
42 http://www.open-std.org/jtc1/sc22/WG9/n412.pdf
43 http://std.dkuug.dk/JTC1/sc22/wg9/n423.pdf
44 http://www.adaic.com/standards/05rm/html/RM-TTL.html

- **Annotated Ada Reference Manual**, ISO/IEC 8652:1995(E) with COR.1:2001 and AMD.1:2007[45] (colored diffs)
- List of Ada Amendment drafts[46]

23.6.5 Ada Issues

- Amendment 200Y[47]
  - AI95-00387-01 Introduction to Amendment ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00387.TXT}
  - AI95-00284-01 New reserved words ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00284.TXT}
  - AI95-00252-01 Object.Operation notation ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00252.TXT}
  - AI95-00218-01 Accidental overloading when overriding ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00218.TXT}
  - AI95-00348-01 Null procedures ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00348.TXT}
  - AI95-00287-01 Limited aggregates allowed ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00287.TXT}
  - AI95-00326-01 Incomplete types ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00326.TXT}
  - AI95-00317-01 Partial parameter lists for formal packages ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00317.TXT}
  - AI95-00376-01 Interfaces.C works for C++ as well ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00376.TXT}
  - AI95-00368-01 Restrictions for obsolescent features ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00368.TXT}
  - AI95-00381-01 New Restrictions identifier No_Dependence ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00381.TXT}
  - AI95-00224-01 pragma Unsuppress ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00224.TXT}
  - AI95-00161-01 Default-initialized objects ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00161.TXT}
  - AI95-00361-01 Raise with message ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00361.TXT}
  - AI95-00286-01 Assert pragma ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00286.TXT}
  - AI95-00328-01 Preinstantiations of Complex_IO ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00328.TXT}
  - AI95-00301-01 Operations on language-defined string types ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00301.TXT}
  - AI95-00340-01 Mod attribute ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00340.TXT}

45 http://www.adaic.com/standards/05aarm/html/AA-TTL.html
46 http://www.ada-auth.org/amendment.html
47 http://www.ada-auth.org/AI-XREF.HTML#Amend_Doc

- AI95-00364-01 Fixed-point multiply/divide ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00364.TXT}
- AI95-00267-01 Fast float-to-integer conversions ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00267.TXT}
- AI95-00321-01 Definition of dispatching policies ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00321.TXT}
- AI95-00329-01 pragma No_Return -- procedures that never return ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00329.TXT}
- AI95-00362-01 Some predefined packages should be recategorized ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00362.TXT}
- AI95-00351-01 Time operations ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00351.TXT}
- AI95-00427-01 Default parameters and Calendar operations ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00427.TXT}
- AI95-00270-01 Stream item size control ^{http://www.ada-auth.org/cgi-bin/cvsweb.cgi/AIs/AI-00270.TXT}

24 Containers

What follows is a simple demo of some of the container types. It does not cover everything, but should get you started.

This language feature is only available in Ada 2005

First Example: Maps

The program below prints greetings to the world in a number of human languages. The greetings are stored in a table, or hashed map. The map associates every greeting (a value) with a language code (a key). That is, you can use language codes as keys to find greeting values in the table.

The elements in the map are constant strings of international characters, or really, pointers to such constant strings. A package `Regional` is used to set up both the language IDs and an instance of Ada .

File: regional.ads

```
with Ada.Containers.Hashed_Maps;  use Ada.Containers;

package Regional is

   type Language_ID is (DE, EL, EN, ES, FR, NL);
   --a selection from the two-letter codes for human languages

   type Hello_Text is access constant Wide_String;
   --objects will contain a «hello»-string in some language


   function ID_Hashed (id: Language_ID) return Hash_Type;
   --you need to provide this to every hashed container

   package Phrases is new Ada.Containers.Hashed_Maps
     (Key_Type => Language_ID,
      Element_Type => Hello_Text,
      Hash => ID_Hashed,
      Equivalent_Keys => "=");

end Regional;
```

Here is the program, details will be explained later.

File: hello_world_extended.ads

```
with Regional; use Regional;
with Ada.Wide_Text_IO; use Ada;

procedure Hello_World_Extended is

   --print greetings in different spoken languages

   greetings: Phrases.Map;
   --the dictionary of greetings

begin  -- Hello_World_Extended

   Phrases.Insert(greetings,
                  Key => EN,
                  New_Item => new Wide_String'("Hello, World!"));

   --or, shorter,
   greetings.Insert(DE, new Wide_String'("Hallo, Welt!"));
   greetings.Insert(NL, new Wide_String'("Hallo, Wereld!"));
   greetings.Insert(ES, new Wide_String'("¡Hola mundo!"));
   greetings.Insert(FR, new Wide_String'("Bonjour, Monde!"));
   greetings.Insert(EL, new Wide_String'("Γειάσου κόσμος"));
                                      --Καλημέρα κόσμε?

   declare
      use Phrases;

      speaker: Cursor := First(greetings);
   begin
      while Has_Element(speaker) loop
         Wide_Text_IO.Put_Line( Element(speaker).all );
         Next(speaker);
      end loop;
   end;

end Hello_World_Extended;
```

The first of the `Insert` statements is written in an Ada 95 style:

```
  Phrases.Insert(greetings,
                 Key => EN,
                 New_Item => new Wide_String'("Hello, World!"));
```

The next insertions use so called distinguished receiver notation which you can use in Ada 2005. (It's O-O parlance. While the Insert call involves all of: a Container object (greetings), a Key object (EN), and a New_Item object (**new** Wide_String'("Hello, World!")), the Container object is distinguished from the others in that the Insert call provides it (and only it) with the other objects. In this case the Container object will be modified by the call, using arguments named Key and New_Item for the modification.)

```
   greetings.Insert(ES, new Wide_String'("¡Hola mundo!"));
```

After the table is set up, the program goes on to print all the greetings contained in the table. It does so employing a cursor that runs along the elements in the table in some order. The typical scheme is to obtain a cursor, here using `First`, and then to iterate the following calls:

1. `Has_Element`, for checking whether the cursor is at an element
2. `Element`, to get the element and
3. `Next`, to move the cursor to another element

When there is no more element left, the cursor will have the special value `No_Element`. Actually, this is an iteration scheme that can be used with all containers in child packages of Ada .

A slight variation: picking an element

The next program shows how to pick a value from the map, given a key. Actually, you will provide the key. The program is like the previous one, except that it doesn't just print all the elements in the map, but picks one based on a Language_ID value that it reads from standard input.

File: hello_world_pick.adb

```
with Regional; use Regional;
with Ada.Wide_Text_IO; use Ada;

procedure Hello_World_Pick is

  ... as before ...

 declare
     use Phrases;

     package Lang_IO is new Wide_Text_IO.Enumeration_IO(Language_ID);
     lang: Language_ID;
  begin
     Lang_IO.Get(lang);
     Wide_Text_IO.Put_Line( greetings.Element(lang).all );
  end;

end Hello_World_Pick;
```

This time the `Element` function consumes a Key (lang) not a Cursor. Actually, it consumes two values, the other value being `greetings`, in distinguished receiver notation.

Second Example: Vectors and Maps

Let's take bean counting literally. Red beans, green beans, and white beans. (Yes, white beans really do exist.) Your job will be to collect a number of beans, weigh them, and then determine the average weight of red, green, and white beans, respectively. Here is one approach.

Again, we need a package, this time for storing vegetable related information. Introducing the `Beans` package (the Grams type doesn't belong in a vegetable package, but it's there to keep things simple):

File: 1/beans.ads

```
with Ada ;

package Beans is

   type Bean_Color is ( R,  G,  W) ;
   --red, green, and white beans

   type Grams is delta 0.  01 digits 7;
   --enough to weigh things as light as beans but also as heavy as
   --many of them

   type Bean is
   --info about a single bean
      record
         kind:   Bean_Color;
         weight:   Grams;
      end record;

   subtype Bean_Count is Positive range 1 ..   1_000;
   --numbers of beans to count (how many has Cinderella have to count?)

   package Bean_Vecs is new Ada.  Containers.  Vectors
     ( Element_Type => Bean,
      Index_Type => Bean_Count) ;

end Beans;
```

The `Vectors` instance offers a data structure similar to an array that can change its size at run time. It is called `Vector`. Each bean that is read will be appended to a `Bean_Vecs.Vector` object.

The following program first calls `read_input` to fill a buffer with beans. Next, it calls a function that computes the average weight of beans having the same color. This function:

```
with Beans;    use Beans;

function average_weight
  ( buffer:   Bean_Vecs.  Vector;  desired_color:   Bean_Color)  return Grams;
--scan `buffer` for all beans that have `desired_color`.  Compute the
--mean of their `.weight` components
```

Then the average value is printed for beans of each color and the program stops.

File: 1/bean_counting.adb

```
with Beans;
with average_weight;
with Ada ;

procedure bean_counting is
   use Beans,  Ada;

   buffer:   Bean_Vecs.  Vector;

   procedure read_input( buf:   in out Bean_Vecs.  Vector)  is separate;
   --collect information from a series of bean measurements into 'buf'


begin --bean_counting

   read_input( buffer) ;

   --now everything is set up for computing some statistical data.
   --For every bean color in 'Bean_Color', the function 'average_weight'
   --will scan 'buffer' once, and accumulate statistical data from
   --each element encountered.

   for kind in Bean_Color loop
      Wide_Text_IO. Put_Line
        ( Bean_Color' Wide_Image( kind)  &
         " ø =" &  Grams' Wide_Image(  average_weight( buffer,  kind)  ) ) ;
   end loop;

end bean_counting;
```

All container operations take place in function `average_weight`. To find the mean weight of beans of the same color, the function is looking at all beans in order. If a bean has the right color, `average_weight` adds its weight to the total weight, and increases the number of beans counted by 1.

The computation visits all beans. The iteration that is necessary for going from one bean to the next and then performing the above steps is best left to the `Iterate` procedure which is part of all container packages. To do so, wrap the above steps inside some procedure and pass this procedure to `Iterate`. The effect is that `Iterate` calls your procedure for each element in the vector, passing a cursor value to your procedure, one for each element.

Having the container machinery do the iteration can also be faster than moving and checking the cursor yourself, as was done in the `Hello_World_Extended` example.

File: average_weight.adb

```
with Beans;   use Beans.  Bean_Vecs;

function average_weight
```

```
   ( buffer:   Bean_Vecs.  Vector;  desired_color:   Bean_Color)  return Grams
is
   total:   Grams := 0.  0;
   --weight of all beans in ‘buffer‘ having ‘desired_color‘

   number:   Natural := 0;
   --number of beans in ‘buffer‘ having ‘desired_color‘

   procedure accumulate( c:   Cursor)  is
      --if the element at ‘c‘ has the ‘desired_color‘, measure it
   begin
      if Element( c) .  kind = desired_color then
         number := number +  1;
         total := total +  Element( c) .  weight;
      end if;
   end accumulate;

begin --average_weight

   Iterate( buffer,  accumulate' Access) ;

   if number >  0 then
      return total /  number;
   else
      return 0.  0;
   end if;

end average_weight;
```

This approach is straightforward. However, imagine larger vectors. `average_weight` will visit all elements repeatedly for each color. If there are M colors and N beans, `average_weight` will be called M * N times, and with each new color, N more calls are necessary. A possible alternative is to collect all information about a bean once it is visited. However, this will likely need more variables, and you will have to find a way to return more than one result (one average for each color), etc. Try it!

A different approach might be better. One is to copy beans of different colors to separate vector objects. (Remembering Cinderella.) Then `average_weight` must visit each element only one time. The following procedure does this, using a new type from `Beans`, called `Bean_Pots`.

```
   ...
   type Bean_Pots is array( Bean_Color)  of Bean_Vecs.  Vector;
   ...
```

Note how this plain array associates colors with Vectors. The procedure for getting the beans into the right bowls uses the bean color as array index for finding the right bowl (vector).

File: 2/gather_into_pots.adb

```
procedure gather_into_pots( buffer:   Bean_Vecs.  Vector;  pots:   in out Bean_Pots)  is
   use Bean_Vecs;

   procedure put_into_right_pot( c:   Cursor)  is
      --select the proper bowl for the bean at ‘c‘ and «append»
```

```
      --the bean to the selected bowl
   begin
      Append( pots( Element( c) .  kind) ,  Element( c) ) ;
   end put_into_right_pot;

begin  --gather_into_pots
   Iterate( buffer,  put_into_right_pot' Access) ;
end gather_into_pots;
```

Everything is in place now.

File: 2/bean_counting.adb

```
with Beans;
with average_weight;
with gather_into_pots;
with Ada.  Wide_Text_IO;

procedure bean_counting is
   use Beans,  Ada;

   buffer:   Bean_Vecs.  Vector;
   bowls:   Bean_Pots;

   procedure read_input( buf:   in out Bean_Vecs.  Vector)  is separate;
   --collect information from a series of bean measurements into ‘buf‘


begin --bean_counting

   read_input( buffer) ;

   --now everything is set up for computing some statistical data.
   --Gather the beans into the right pot by color.
   --Then find the average weight of beans in each pot.

   gather_into_pots( buffer,  bowls) ;

   for color in Bean_Color loop
      Wide_Text_IO. Put_Line
        ( Bean_Color' Wide_Image( color)
         & " ø ="
         & Grams' Wide_Image( average_weight( bowls( color) ,  color) ) ) ;
   end loop;

end bean_counting;
```

As a side effect of having chosen one vector per color, we can determine the number of beans in each vector by calling the `Length` function. But `average_weight`, too, computes the number of elements in the vector. Hence, a summing function might replace `average_weight` here.

All In Just One Map!

The following program first calls `read_input` to fill a buffer with beans. Then, information about these beans is stored in a table, mapping bean properties to numbers of occurrence.

The processing that starts at `Iterate` uses chained procedure calls typical of the Ada iteration mechanism.

The Beans package in this example instantiates another generic library unit, Ada . Where the Ada require a hashing function, Ada require a comparison function. We provide one, `"<"`, which sorts beans first by color, then by weight. It will automatically be associated with the corresponding generic formal function, as its name, `"<"`, matches that of the generic formal function, `"<"`.

```
...
function "<"( a,  b:   Bean)  return Boolean;
--order beans, first by color, then by weight

package Bean_Statistics
  --instances will map beans of a particular color and weight to the
  --number of times they have been inserted.
is new Ada.  Containers.  Ordered_Maps
  ( Element_Type => Natural,
   Key_Type => Bean) ;
...
```

Where the previous examples have **with**ed subprograms, this variation on `bean_counting` packs them all as local subprograms.

File: 3/bean_counting.adb

```
with Beans;
with Ada.  Wide_Text_IO;

procedure bean_counting is
   use Beans,  Ada;

   buffer:   Bean_Vecs.  Vector;
   stats_cw:   Bean_Statistics.  Map;
   --maps beans to numbers of occurrences, grouped by color, ordered by
   --weight

   procedure read_input( buf:   in out Bean_Vecs.  Vector)  is separate;
   --collect information from a series of bean measurements into `buf`

   procedure add_bean_info( specimen:   in Bean) ;
   --insert bean `specimen` as a key into the `stats_cw` table unless
   --present.  In any case, increase the count associated with this key
   --by 1.  That is, count the number of equal beans.

   procedure add_bean_info( specimen:   in Bean)  is

      procedure one_more( b:   in Bean;  n:   in out Natural)  is
       --increase the count associated with this kind of bean
      begin
         n := n +  1;
      end one_more;

      c :   Bean_Statistics.  Cursor;
      inserted:   Boolean;
   begin
      stats_cw.  Insert( specimen,  0,  c,  inserted) ;
      Bean_Statistics.  Update_Element( c,  one_more' Access) ;
```

```
   end add_bean_info;

begin --bean_counting

   read_input( buffer) ;

   --next, for all beans in the vector ‘buffer‘ just filled, store
   --information about each bean in the ‘stats_cw‘ table.

   declare
      use Bean_Vecs;

      procedure count_bean( c:   Cursor)  is
      begin
         add_bean_info( Element( c) ) ;
      end count_bean;
   begin
      Iterate( buffer,  count_bean' Access) ;
   end;

   --now everything is set up for computing some statistical data.  The
   --keys of the map, i.e.  beans, are ordered by color and then weight.
   --The ‘First‘, and ‘Ceiling‘ functions will find cursors
   --denoting the ends of a group.


   declare
      use Bean_Statistics;

      --statistics is computed groupwise:

      q_sum:   Grams;
      q_count:   Natural;

      procedure q_stats( lo,  hi:   Cursor) ;
      --‘q_stats‘ will update the ‘q_sum‘ and ‘q_count‘ globals with
      --the sum of the key weights and their number, respectively.  ‘lo‘
      --(included) and ‘hi‘ (excluded) mark the interval of keys
      --to use from the map.

      procedure q_stats( lo,  hi:   Cursor)  is
         k:   Cursor := lo;
      begin
         q_count := 0;  q_sum := 0.  0;
         loop
            exit when k = hi;
            q_count := q_count +  Element( k) ;
            q_sum := q_sum +  Key( k) .  weight *  Element( k) ;
            Next( k) ;
         end loop;
      end q_stats;


      --precondition
      pragma assert( not Is_Empty( stats_cw) ,  "container is empty") ;

      lower,  upper:   Cursor := First( stats_cw) ;
      --denoting the first key of a group, and the first key of a
      --following group, respectively

   begin

      --start reporting and trigger the computations

      Wide_Text_IO. Put_Line( "Summary:") ;
```

```
      for color in Bean_Color loop
         lower := upper;
         if color = Bean_Color' Last then
            upper := No_Element;
         else
            upper := Ceiling( stats_cw,  Bean' ( Bean_Color' Succ( color) ,
0.  0) ) ;
         end if;

         q_stats( lower,  upper) ;

         if q_count >  0 then
            Wide_Text_IO. Put_Line
              ( Bean_Color' Wide_Image( color)  & " group:" &
               "  ø =" & Grams' Wide_Image( q_sum /  q_count)  &
               ", # =" & Natural' Wide_Image( q_count)  &
               ", Σ =" & Grams' Wide_Image( q_sum) ) ;
         end if;
      end loop;
   end;

end bean_counting;
```

Like in the greetings example, you can pick values from the table. This time the values tell the number of occurrences of beans with certain properties. The `stats_cw` table is ordered by key, that is by bean properties. Given particular properties, you can use the `Floor` and `Ceiling` functions to approximate the bean in the table that most closely matches the desired properties.

It is now easy to print a histogram showing the frequency with which each kind of bean has occurred. If N is the number of beans of a kind, then print N characters on a line, or draw a graphical bar of length N, etc. A histogram showing the number of beans per color can be drawn after computing the sum of beans of this color, using groups like in the previous example. You can delete beans of a color from the table using the same technique.

Finally, think of marshalling the beans in order starting at the least frequently occurring kind. That is, construct a vector appending first beans that have occurred just once, followed by beans that have occurred twice, if any, and so on. Starting from the table is possible, but be sure to have a look at the sorting functions of Ada .

24.1 See also

See also

24.1.1 Wikibook

- Ada Programming[1]
- Ada Programming/Libraries/Ada.Containers[2]

1 http://en.wikibooks.org/wiki/Ada%20Programming
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers

24.1.2 Ada 2005 Reference Manual

- A.18.1 The Package Containers ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-18-1.html}

25 Interfacing

25.1 Interfacing

Interfacing

Ada is one of the few languages where interfacing is part of the language standard. The programmer can interface with other programming languages, or with the hardware.

25.2 Other programming languages

Other programming languages

The language standard defines the interfaces for C[1], Cobol[2] and Fortran[3]. Of course any implementation might define further interfaces — GNAT[4] for example defines an interface to C++[5].

Interfacing with other languages is actually provided by pragma Export[6], Import[7] and Convention[8].

25.3 Hardware devices

Hardware devices

Embedded programmers usually have to write device drivers. Ada provides extensive support for interfacing with hardware, like using representation clauses[9] to specify the exact representation of types used by the hardware, or standard interrupt handling for writing Interrupt service routine[10]s.

1 http://en.wikibooks.org/wiki/C%20Programming
2 http://en.wikibooks.org/wiki/COBOL
3 http://en.wikibooks.org/wiki/Programming%3AFortran
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FGNAT
5 http://en.wikibooks.org/wiki/C%2B%2B%20Programming
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FConvention
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FRepresentation%20clauses
10 http://en.wikipedia.org/wiki/Interrupt%20service%20routine

25.4 See also

See also

25.4.1 Wikibook

- Ada Programming[11]
- Ada Programming/Libraries/Interfaces[12]

25.4.2 Ada Reference Manual

- Annex B Interface to Other Languages ˆ{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B.html}
- Annex C Systems Programming ˆ{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-C.html}

25.4.3 Ada 95 Rationale

- bInterface to Other Languages3

25.4.4 Ada Quality and Style Guide

- 7.6.4 Interfacing to Foreign Languages ˆ{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-6-4.html}

11 http://en.wikibooks.org/wiki/Ada%20Programming
12 Chapter 43 on page 349

26 Coding Standards

26.1 Introduction

Introduction

Each project should follow a specific coding standard[1] to ease readability and maintenance of the source code, and reduce the insertion of errors. Depending on the requirements of the project, a set of guidelines can help to achieve the desired level of performance, portability, code complexity...

There are many ASIS[2] tools that can be used to check automatically the adherence of Ada source code to the guidelines.

26.2 Tools

Tools

- AdaControl[3] (Rules[4])
- gnatcheck[5] (Rules[6])
- GNAT Pretty-Printer[7]
- The GNAT Metric Tool gnatmetric[8]
- RainCode Engine[9]
- RainCode Checker[10]
- AdaSTAT[11]

26.3 Coding guidelines

1 http://en.wikipedia.org/wiki/coding%20standard
2 http://en.wikipedia.org/wiki/Ada%20Semantic%20Interface%20Specification
3 http://www.adalog.fr/adacontrol2.htm
4 http://www.adalog.fr/compo/adacontrol_ug.html#Rules-reference
5 http://gcc.gnu.org/onlinedocs/gnat_ugn_unw/Verifying-Properties-Using-gnatcheck.html
6 http://gcc.gnu.org/onlinedocs/gnat_ugn_unw/Predefined-Rules.html
7 http://gcc.gnu.org/onlinedocs/gnat_ugn_unw/The-GNAT-Pretty_002dPrinter-gnatpp.html
8 http://gcc.gnu.org/onlinedocs/gnat_ugn_unw/The-GNAT-Metric-Tool-gnatmetric.html
9 http://www.raincode.com/adaengine.html
10 http://www.raincode.com/adachecker.html
11 http://www.adastat.com/

Coding guidelines

- *Ada Quality & Style Guide*[12]*: Guidelines for Professional Programmers*
- ISO/IEC TR 15942:2000, *Guide for the use of the Ada programming language in high integrity systems*[13], First edition (2000-03-01). ISO Freely Available Standards[14]
- Stephen Leake, *NASA Flight Software Branch — Ada Coding Standard*[15] (2004-01-30)
- ESA[16] BSSC

```
    |      title = Ada Coding Standard
```

. , ,

```
  |    edition = BSSC(98)3 Issue 1
  |       year = 1998
  |      month = October
  |        url = ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/bssc983.pdf
  | accessdate = 2009-01-19
```

. , ,

- GNAT Coding Style: A Guide for GNAT Developers

```
    |       work = GCC online documentation
```

[17]. . Retrieved

```
  |  publisher = Free Software Foundation
  |        url = http://gcc.gnu.org/onlinedocs/gnat-style/
  | accessdate = 2009-01-19
```

[18]. . Retrieved (PDF[19])

26.4 See also

See also

26.4.1 Other wikibooks

- Ada Style Guide[20]

12 http://en.wikibooks.org/wiki/Ada%20Style%20Guide
13 http://www.dit.upm.es/ork/documents/adahis.pdf
14 http://standards.iso.org/ittf/PubliclyAvailableStandards/
15 http://software.gsfc.nasa.gov/AssetsApproved/PA2.4.1.1.1.pdf
16 http://en.wikipedia.org/wiki/European%20Space%20Agency
17
18
19 http://gcc.gnu.org/onlinedocs/gnat-style.pdf
20 http://en.wikibooks.org/wiki/Ada%20Style%20Guide

26.4.2 Wikibook

- Ada Programming[21]

26.4.3 Ada Quality and Style Guide

- Chapter 1: Introduction ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_1/}

26.5 External links

External links

- Introduction to Coding Standards[22]

21 http://en.wikibooks.org/wiki/Ada%20Programming

22 http://geekswithblogs.net/sdorman/archive/2007/06/13/Introduction-to-Coding-Standards.aspx

27 Tips

27.1 Full declaration of a type can be deferred to the unit's body

Full declaration of a type can be deferred to the unit's body

Often, you'll want to make changes to the internals of a private type. This, in turn, will require the algorithms that act on it to be modified. If the type is completed in the unit specification, it is a pain to edit and recompile both files, even with an IDE[1], but it's something some programmers learn to live with.

It turns out you don't have to. Nonchalantly mentioned in the ARM[2], and generally skipped over in tutorials, is the fact that private types can be completed in the unit's body itself, making them much closer to the relevant code, and saving a recompile of the specification, as well as every unit depending on it. This may seem like a small thing, and, for small projects, it is. However, if you have one of those uncooperative types that requires dozens of tweaks, or if your dependence graph has much depth, the time and annoyance saved add up quickly.

Also, this construction is very useful when coding a shared library, because it permits to change the implementation of the type while still providing a compatible ABI[3].

Code sample:

```
package Private_And_Body is

  type Private_Type is limited private;

  --Operations...

private
  type Body_Type;   --Defined in the body
  type Private_Type is access Body_Type;
end Private_And_Body;
```

The type in the public part is an access[4] to the hidden type. This has the drawback that memory management has to be provided by the package implementation. That is the reason why `Private_Type` is a limited type, the client will not be allowed to copy the access values, in order to prevent dangling references.

1 http://en.wikipedia.org/wiki/Integrated%20development%20environment
2 http://www.adaic.org/standards/95lrm/html/RM-TTL.html
3 http://en.wikipedia.org/wiki/Application_binary_interface
4 Chapter 13 on page 99

These types are sometimes called "Taft types" —named after Tucker Taft, the main designer of Ada 95— because were introduced in the so-called Taft Amendment to Ada 83. In other programming languages, this technique is called "opaque pointer[5]s".

27.2 Lambda calculus through generics

Lambda calculus through generics

Suppose you've decided to roll your own set[6] type. You can add things to it, remove things from it, and you want to let a user apply some arbitrary function to all of its members. But the scoping rules seem to conspire against you, forcing nearly everything to be global.

The mental stumbling block is that most examples given of generics[7] are packages, and the Set package is already generic. In this case, the solution is to make the Apply_To_All procedure generic as well; that is, to nest the generics. Generic procedures inside packages exist in a strange scoping limbo, where anything in scope at the instantiation can be used by the instantiation, and anything normally in scope at the formal can be accessed by the formal. The end result is that the relevant scoping roadblocks no longer apply. It isn't the full lambda calculus, just one of the most useful parts.

```
generic
  type Element is private;
package Sets is
  type Set is private;
   [..]
  generic
    with procedure Apply_To_One (The_Element : in out Element);
  procedure Apply_To_All (The_Set : in out Set);
end Sets;
```

For a view of Functional Programming in Ada see [8].

27.3 Compiler Messages

Compiler Messages

Different compilers can diagnose different things differently, or the same thing using different messages, etc.. Having two compilers at hand can be useful.

`selected component`

When a source program contains a construct such as `Foo.Bar`, you may see messages saying something like «selected component "Bar"» or maybe like «selected component "Foo"». The phrases may seem confusing, because one refers to `Foo`, while the other refers

5 `http://en.wikipedia.org/wiki/opaque%20pointer`
6 `http://en.wikipedia.org/wiki/Set%20%28computer%20science%29`
7 Chapter 20 on page 161
8 Functional Programming in...Ada? ^{`http://okasaki.blogspot.com/2008/07/functional-programming-inada.html`} , by Chris Okasaki

to `Bar`. But they are both right. The reason is that `selected_component` is an item from Ada's grammar (4.1.3 Selected Components ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-1-3.html`}). It denotes all of: a prefix, a dot, and a selector_name. In the `Foo.Bar` example these correspond to `Foo`, `'.'`, and `Bar`. Look for more grammar words in the compiler messages, e.g. «prefix», and associate them with identifiers quoted in the messages.

For example, if you submit the following code to the compiler,

```
with Pak;
package Foo is
   type T is new Pak.  Bar;   --Oops, Pak is generic!
end Foo;
```

the compiler may print a diagnostic message about a prefixed component: `Foo`'s author thought that `Pak` denotes a package, but actually it is the name of a *generic* package. (Which needs to be instantiated first; and then the *instance* name is a suitable prefix.)

27.4 Universal integers

Universal integers

All integer literals and also some attributes like `'Length` are of the anonymous type *universal_integer*, which comprises the infinite set of mathematical integers. Named numbers are of this type and are evaluated exactly (no overlow except for machine storage limitations), e.g.

```
Very_Big: constant := 10**1_000_000 - 1;
```

Since *universal_integer* has no operators, its values are converted in this example to *root_integer*, another anonymous type, the calcuation is performed and the result again converted back in *universal_integer*.

Generally values of *universal_integer* are implicitly converted to the appropriate type when used in some expression. So the expression **not** `A' Length` is fine; the value of `A' Length` is interpreted as a modular integer since **not** can only be applied to modular integers (of course a context is needed to decide which modular integer type is meant). This feature can lead to pitfalls. Consider

```
type Ran_6 is range 1 ..   6;
type Mod_6 is mod 6;
```

and then

```
if A' Length in Ran_6 then  --OK
   ...

if not A' Length in Ran_6 then  --not OK
```

```
   ...
--this is the same as
if (not A' Length) in Ran_6 then  --not OK
   ...

if A' Length in 1 ..   6 then  --OK
   ...

if not A' Length in 1 ..   6 then  --not OK
   ...

if A' Length in Mod_6 then  --OK?
   ...

if not A' Length in Mod_6 then  --OK?
   ...
```

The second conditional cannot be compiled because the expressions to the left of **in** is incompatible to the type at the right. Note that **not** has precedence over **in**. It does not negate the entire membership test but only `A' Length`.

The fourth conditional fails in various ways.

The sixth conditional might be fine because **not** turns `A' Length` into a modular value which is OK if the value is covered by modular type `Mod_6`.

GNAT GPL 2009 gives these diagnoses respectively:

```
error: incompatible types
error: operand of not must be enclosed in parentheses
warning: not expression should be parenthesized here
```

A way to *avoid* these problems is to use **not in** for the membership test,

```
   if A' Length not in Ran_6 then  --OK
   ...
```

See

- 2.4 Numeric Literals ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-4.html} ,
- 3.6.2 Operations of Array Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-6-2.html}), and
- 4.5 Operators and Expression Evaluation ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-5.html} ,
- 4.5.2 Relational Operators and Membership Tests ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-5-2.html} ,
- Membership Tests[9]

27.5 I/O

9 Chapter 37.3 on page 303

27.5.1 Text_IO Issues

A canonical method of reading a sequence of lines from a text file uses the standard procedure Ada .*Get_Line*. When the end of input is reached, *Get_Line* will fail, and exception *End_Error* is raised. Some programs will use another function from Ada to prevent this and test for *End_of_Input*. However, this isn't always the best choice, as has been explained for example in a Get_Line news group discussion on comp.lang.ada[10].

A working solution uses an exception handler instead:

```
declare
   The_Line:   String( 1..  100) ;
   Last:   Natural;
begin
   loop
      Text_IO. Get_Line( The_Line,  Last) ;
      --do something with The_Line ...
   end loop;
exception
   when Text_IO. End_Error =>
      null;
end;
```

27.6 Quirks

Quirks

Using GNAT on Windows, calls to subprograms from Ada might need special attention. (For example, the `Real_Time.Clock` function might seem to return values indicating that no time has passed between two invocations when certainly some time has passed.) The cause is reported to be a missing initialization of the run-time support when no other real-time features are present in the program.[11] As a provisional fix, it is suggested to insert

```
delay 0.  0;
```

before any use of `Real_Time` services.

27.6.1 Stack Size

With some implementations, notably GNAT, knowledge of stack size manipulation will be to your advantage. Executables produced with GNAT tools and standard settings can hit the stack size limit. If so, the operating system might allow setting higher limits. Using

10 `http://groups.google.com/group/comp.lang.ada/browse_thread/thread/5afe598156615c8b#`

11 Vincent Celier . Timing code blocks Timing code blocks ^{`groups.google.es/group/comp.lang.ada/browse_thread/thread/c8acfc87fbb1813d`} . , Usenet article forwards this information from AdaCore.

GNU/Linux and the Bash command shell, try

```
$ ulimit -s [some number]
```

The current value is printed when only `-s` is given to *ulimit*.

27.7 References

References

27.8 See also

See also

27.8.1 Wikibook

- Ada Programming[12]
- Ada Programming/Errors[13]

27.8.2 Ada Reference Manual

- 3.10.1 Incomplete Type Declarations ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-10-1.html`}

12 `http://en.wikibooks.org/wiki/Ada%20Programming`
13 Chapter 28 on page 255

28 Common Errors

Some language features are often misunderstood, resulting in common programming errors, performance degradation and portability problems. The following incorrect usages of the Ada language are often seen in code written by Ada beginners.

28.1 pragma Atomic & Volatile

pragma Atomic & Volatile

It **is almost always incorrect to use atomic[1] or volatile[2] variables for tasking[3]**.[4] When an object is atomic it just means that it will be read from or written to memory atomically. The compiler *will not* generate atomic instructions or memory barriers when accessing to that object, it will just:

- check that the architecture guarantees atomic memory loads and stores,
- disallow some compiler optimizations, like reordering or suppressing redundant accesses to the object.

For example, the following code, where `A` is an atomic object can be misunderstood:

```
A := A + 1;  --Not an atomic increment!
```

The compiler **will not** (and is not allowed by the Standard to) generate an atomic increment instruction to directly increment and update from memory the variable `A`.[6] This is the code generated by the compiler:

```
  A := A + 1;
804969f:    a1 04 95 05 08          mov    0x8059504,%eax
80496a4:    40                      inc    %eax
80496a5:    a3 04 95 05 08          mov    %eax,0x8059504
```

As can be seen, no atomic increment instruction or test-and-set opcode will be generated. Like in other programming languages, if these specific instructions are required in the program they must be written explicitly using machine code insertions.[8]

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAtomic
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FVolatile
3 Chapter 21 on page 173
4 Volatile: Almost Useless for Multi-Threaded Programming [5]. Intel Software Network . Retrieved 2008-05-30
6 Volatile [7]. . Retrieved 2008-05-28
8 Laurent Guerby Ada 95 Rationale . Intermetrics , , 1995

The above code snippet is equivalent to the following code (both code sequences generates exactly the same object code), where `T` is a (non-atomic) temporary variable:

```
T := A;      --A is copied atomically to local variable T
T := T + 1;  --local variable T is incremented
A := T;      --A is stored atomically
```

Thus it is incorrect to modify an atomic variable at the same time from multiple tasks. For example, two tasks incrementing a counter in parallel. Even in an uniprocessor, other Ada tasking features like a protected object should be used instead. In multiprocessors, depending on the memory consistency model[9], using various atomic or volatile variables for task communication can have surprising consequences.[10][12] Therefore, extreme care should be taken when using atomic objects for task data sharing or synchronization, specially in a multiprocessor.

28.2 References

References

28.3 pragma Pack

pragma Pack

28.3.1 Exact data representation

It is important to realize that **pragma Pack[13] must not be used to specify the exact representation of a data type**, but to help the compiler to improve the efficiency of the generated code.[14] The compiler is free to ignore the pragma, therefore if a specific representation of a type is required, representation clauses[15] should be used instead (record representation clauses, and/or attributes 'Size[16] or 'Component_Size[17]).

---

9 http://en.wikipedia.org/wiki/Memory%20model%20%28programming%29

10 Volatile [11]. . Retrieved 2008-05-28

12 Sarita V. Adve, Kourosh Gharachorloo . Shared Memory Consistency Models: A Tutorial Shared Memory Consistency Models: A Tutorial ^{www.hpl.hp.com/techreports/Compaq-{}DEC/WRL-{}95-{}7.pdf} . *IEEE Computer* , **29** : 66−76 December 1996

13 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPack

14 Adam Beneschan . Pragma Pack vs. Convention C, portability issue? Pragma Pack vs. Convention C, portability issue? ^{groups.google.es/group/comp.lang.ada/msg/6698960624779ec7} . ,

15 http://en.wikibooks.org/wiki/Ada%20Programming%2FRepresentation%20clauses

16 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Size

17 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Component_Size

28.3.2 Bit-wise operations

Although in Ada 83 packed boolean arrays were used for bit-wise operations,[18] since Ada 95 modular types[19] are more adequate for these operations.[20] The argument may be weighed against the advantages of named Boolean array indexes such as `Traffic_Lights' ( Red => True, others => False)` , depending on use case.

28.4 'Bit_Order attribute

'Bit_Order attribute

The **'Bit_Order[21] attribute is not intended to convert data between a big-endian and a little-endian machine** (it affects bit numbering, not byte order). The compiler will not generate code to reorder multi-byte fields when a non-native bit order is specified.[22][23][24]

28.5 'Size attribute

'Size attribute

A common Ada programming mistake is to assume that specifying 'Size for a type T forces the compiler to allocate exactly this number of bits for objects of this type. This is not true. **The specified T'Size[26] will force the compiler to use this size for components in packed arrays and records and in Unchecked_Conversion**, but the compiler is still free to allocate more bits for stand-alone objects.

Use 'Size on the object itself to force the object to the specified value.

28.6 See also

18 Software Productivity Consortium (October 1995). *Ada 95 Quality and Style Guide*, "10.5.7 Packed Boolean Array Shifts ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_10/10-5-7.html} "

19 Chapter 7 on page 71

20 Software Productivity Consortium (October 1995). *Ada 95 Quality and Style Guide*, "10.6.3 Bit Operations on Modular Types ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_10/10-6-3.html} "

21 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Bit%20Order

22 . . ,

23 ISO/IEC 8652:2007. 13.5.3 Bit Ordering (9/2). Ada 2005 Reference Manual. Bit_Order clauses make it possible to write record_representation_clauses that can be ported between machines having different bit ordering. They do not guarantee transparent exchange of data between such machines. ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-13-5-3.html}

24 [25]. . Retrieved

26 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Size

See also

28.6.1 Wikibook

- Ada Programming[27]
- Ada Programming/Tips[28]

28.7 References

References

27 http://en.wikibooks.org/wiki/Ada%20Programming
28 Chapter 27 on page 249

29 Algorithms

29.1 Introduction

Introduction

Welcome to the Ada implementations of the Algorithms[1] Wikibook. For those who are new to Ada Programming[2] a few notes:

- All examples are fully functional with all the needed input and output operations. However, only the code needed to outline the algorithms at hand is copied into the text - the full samples are available via the download links. (Note: It can take up to 48 hours until the cvs is updated).
- We seldom use predefined types in the sample code but define special types suitable for the algorithms at hand.
- Ada allows for default function parameters; however, we always fill in and name all parameters, so the reader can see which options are available.
- We seldom use shortcuts - like using the attributes *Image* or *Value* for String <=> Integer conversions.

All these rules make the code more elaborate than perhaps needed. However, we also hope it makes the code easier to understand

Category:Ada Programming[3]

29.2 Chapter 1: Introduction

Chapter 1: Introduction

The following subprograms are implementations of the *Inventing an Algorithm* examples[4].

29.2.1 To Lower

The Ada example code does not append to the array as the algorithms. Instead we create an empty array of the desired length and then replace the characters inside.

1 http://en.wikibooks.org/wiki/Algorithms
2 http://en.wikibooks.org/wiki/Ada%20Programming
3 http://en.wikibooks.org/wiki/Category%3AAda%20Programming
4 http://en.wikibooks.org/wiki/Algorithms%2FIntroduction%23Inventing%20an%20Algorithm

File: to_lower_1.adb

```
function To_Lower (C : Character) return Character renames
   Ada.Characters.Handling.To_Lower;

--  tolower - translates all alphabetic, uppercase characters
--  in str to lowercase
function To_Lower (Str : String) return String is
   Result : String (Str'Range);
begin
   for C in  Str'Range loop
      Result (C) := To_Lower (Str (C));
   end loop;
   return Result;
end To_Lower;
```

Would the append approach be impossible with Ada? No, but it would be significantly more complex and slower.

29.2.2 Equal Ignore Case

File: to_lower_2.adb

```
--  equal-ignore-case -- returns true if s or t are equal,
--  ignoring case
function Equal_Ignore_Case
  (S    : String;
   T    : String)
   return Boolean
is
   O : constant Integer := S'First - T'First;
begin
   if T'Length /= S'Length then
      return False;  --  if they aren't the same length, they
                     --  aren't equal
   else
      for I in  S'Range loop
         if To_Lower (S (I)) /=
            To_Lower (T (I + O))
         then
            return False;
         end if;
      end loop;
   end if;
   return True;
end Equal_Ignore_Case;
```

29.3 Chapter 6: Dynamic Programming

29.3.1 Fibonacci numbers

The following codes are implementations of the Fibonacci-Numbers examples[5].

Simple Implementation

File: fibonacci_1.adb

```
...
```

To calculate Fibonacci numbers negative values are not needed so we define an integer type which starts at 0. With the integer type defined you can calculate up until `Fib (87)`. `Fib (88)` will result in an `Constraint_Error`.

```
   type Integer_Type is range 0 .. 999_999_999_999_999_999;
```

You might notice that there is not equivalence for the `assert (n >= 0)` from the original example. Ada will test the correctness of the parameter *before* the function is called.

```
   function Fib (n : Integer_Type) return Integer_Type is
   begin
      if n = 0 then
         return 0;
      elsif n = 1 then
         return 1;
      else
         return Fib (n - 1) + Fib (n - 2);
      end if;
   end Fib;

...
```

Cached Implementation

File: fibonacci_2.adb

```
...
```

For this implementation we need a special cache type can also store a -1 as "not calculated" marker

5 http://en.wikibooks.org/wiki/Algorithms%2FDynamic%20Programming%23Fibonacci_Numbers

```
   type Cache_Type is range -1 .. 999_999_999_999_999_999;
```

The actual type for calculating the fibonacci numbers continues to start at 0. As it is a **subtype** of the cache type Ada will automatically convert between the two. (the conversion is - of course - checked for validity)

```
   subtype Integer_Type is Cache_Type range
      0 .. Cache_Type'Last;
```

In order to know how large the cache need to be we first read the actual value from the command line.

```
   Value : constant Integer_Type :=
      Integer_Type'Value (Ada.Command_Line.Argument (1));
```

The Cache array starts with element 2 since Fib (0) and Fib (1) are constants and ends with the value we want to calculate.

```
   type Cache_Array is
      array (Integer_Type range 2 .. Value) of Cache_Type;
```

The Cache is initialized to the first valid value of the cache type — this is `-1`.

```
   F : Cache_Array := (others => Cache_Type'First);
```

What follows is the actual algorithm.

```
   function Fib (N : Integer_Type) return Integer_Type is
   begin
      if N = 0 or else N = 1 then
         return N;
      elsif F (N) /= Cache_Type'First then
         return F (N);
      else
         F (N) := Fib (N - 1) + Fib (N - 2);
         return F (N);
      end if;
   end Fib;

...
```

This implementation is faithful to the original from the Algorithms[6] book. However, in Ada you would normally do it a little different:

File: fibonacci_3.adb

6 http://en.wikibooks.org/wiki/Algorithms

when you use a slightly larger array which also stores the elements 0 and 1 and initializes them to the correct values

```
type Cache_Array is
   array (Integer_Type range 0 .. Value) of Cache_Type;

F : Cache_Array :=
   (0      => 0,
    1      => 1,
    others => Cache_Type'First);
```

and then you can remove the first **if** path.

```
      return N;
   els
```

```
if F (N) /= Cache_Type'First then
```

This will save about 45% of the execution-time (measured on Linux i686) while needing only two more elements in the cache array.

Memory Optimized Implementation

This version looks just like the original in WikiCode.

File: fibonacci_4.adb

```
type Integer_Type is range 0 .. 999_999_999_999_999_999;

function Fib (N : Integer_Type) return Integer_Type is
   U : Integer_Type := 0;
   V : Integer_Type := 1;
begin
   for I in  2 .. N loop
      Calculate_Next : declare
         T : constant Integer_Type := U + V;
      begin
         U := V;
         V := T;
      end Calculate_Next;
   end loop;
   return V;
end Fib;
```

No 64 bit integers

Your Ada compiler does not support 64 bit integer numbers? Then you could try to use decimal numbers[7] instead. Using decimal numbers results in a slower program (takes about three times as long) but the result will be the same.

The following example shows you how to define a suitable decimal type. Do experiment with the **digits** and **range** parameters until you get the optimum out of your Ada compiler.

File: fibonacci_5.adb

```
type Integer_Type is delta 1.0 digits 18 range
   0.0 .. 999_999_999_999_999_999.0;
```

You should know that floating point numbers are unsuitable for the calculation of fibonacci numbers. They will not report an error condition when the number calculated becomes too large — instead they will lose in precision which makes the result meaningless.

7 Chapter 10 on page 79

30 Function overloading

File: function_overloading.adb

```
function Generate_Number (MaxValue : Integer) return Integer is
   subtype Random_Type is Integer range 0 .. MaxValue;
   package Random_Pack is new Ada.Numerics.Discrete_Random  (Random_Type);

   G : Random_Pack.Generator;
begin
   Random_Pack.Reset (G);
   return Random_Pack.Random (G);
end Generate_Number;


function Generate_Number (MinValue : Integer;
                          MaxValue : Integer) return Integer
is
   subtype Random_Type is Integer range MinValue .. MaxValue;
   package Random_Pack is new Ada.Numerics.Discrete_Random  (Random_Type);

   G : Random_Pack.Generator;
begin
   Random_Pack.Reset (G);
   return Random_Pack.Random (G);
end Generate_Number;
```

```
Number_1 : Integer := Generate_Number (10);
```

```
Number_2 : Integer := Generate_Number (6, 10);
```

30.1 Function overloading in Ada

Function overloading in Ada

Ada supports all six signature options but if you use the arguments' name as option you will always have to name the parameter when calling the function. i.e.:

```
Number_2 : Integer := Generate_Number (MinValue => 6,
                                       MaxValue => 10);
```

Note that you cannot overload a generic procedure or generic function within the same package. The following example will fail to compile:

```
package myPackage
  generic
    type Value_Type is (<>);
  --The first declaration of a generic subprogram
  --with the name "Generic_Subprogram"
  procedure Generic_Subprogram (Value : in out Value_Type);
  ...
  generic
    type Value_Type is (<>);
  --This subprogram has the same name, but no
  --input or output parameters.  A non-generic
  --procedure would be overloaded here.
  --Since this procedure is generic, overloading
  --is not allowed and this package will not compile.
  procedure Generic_Subprogram;
  ...
  generic
    type Value_Type is (<>);
  --The same situation.
  --Even though this is a function and not
  --a procedure, generic overloading of
  --the name "Generic_Subprogram" is not allowed.
  function Generic_Subprogram (Value : Value_Type) return Value_Type;
end myPackage;
```

30.2 See also

See also

30.2.1 Wikibook

- Ada Programming[1]
- Ada Programming/Subprograms[2]

30.2.2 Ada 95 Reference Manual

- 6.6 Overloading of Operators ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-6-6.html}
- 8.6 The Context of Overload Resolution ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-8-6.html}

30.2.3 Ada 2005 Reference Manual

- 6.6 Overloading of Operators ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-6-6.html}
- 8.6 The Context of Overload Resolution ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-8-6.html}

1 http://en.wikibooks.org/wiki/Ada%20Programming
2 Chapter 16 on page 125

31 Mathematical calculations

Ada is very well suited for all kind of calculations. You can define you own fixed point and floating point types and — with the aid of generic packages call all the mathematical functions you need. In that respect Ada is on par with Fortran[1]. This module will show you how to use them and while we progress we create a simple RPN[2] calculator.

31.1 Simple calculations

Simple calculations

31.1.1 Addition

Additions can be done using the predefined operator + . The operator is predefined for all numeric types and the following, working code, demonstrates its use:

File: numeric_1.adb

```
-- The Package Text_IO[3]
with Ada ;

procedure Numeric_1 is
   type Value_Type is digits 12
         range -999_999_999_999.0e999 .. 999_999_999_999.0e999;

   package T_IO renames Ada.Text_IO;
   package F_IO is new  Ada.Text_IO.Float_IO (Value_Type);

   Value_1 : Value_Type;
   Value_2 : Value_Type;

begin
   T_IO.Put ("First Value : ");
   F_IO.Get (Value_1);
   T_IO.Put ("Second Value : ");
   F_IO.Get (Value_2);

   F_IO.Put (Value_1);
   T_IO.Put (" + ");
   F_IO.Put (Value_2);
   T_IO.Put (" = ");
   F_IO.Put (Value_1 +  Value_2);
end Numeric_1;
```

1 http://en.wikibooks.org/wiki/Programming%3AFortran
2 http://en.wikipedia.org/wiki/Reverse%20Polish%20notation

31.1.2 Subtraction

Subtractions can be done using the predefined operator - . The following extended demo shows the use of + and - operator together:

File: numeric_2.adb

```
-- The Package Text_IO[4]
with Ada ;

procedure Numeric_2
is
  type Value_Type
  is digits
     12
  range
     -999_999_999_999.0e999 .. 999_999_999_999.0e999;

  package T_IO renames Ada ;
  package F_IO is new  Ada.Text_IO.Float_IO (Value_Type);

  Value_1   : Value_Type;
  Value_2   : Value_Type;
  Result    : Value_Type;
  Operation : Character;

begin
  T_IO.Put ("First Value  : ");
  F_IO.Get (Value_1);
  T_IO.Put ("Second Value : ");
  F_IO.Get (Value_2);
  T_IO.Put ("Operation    : ");
  T_IO.Get (Operation);

  case Operation is
     when '+' =>
        Result := Value_1 + Value_2;
     when '-' =>
        Result := Value_1 - Value_2;
     when others =>
        T_IO.Put_Line ("Illegal Operation.");
        goto Exit_Numeric_2;
  end case;

  F_IO.Put (Value_1);
  T_IO.Put (" ");
  T_IO.Put (Operation);
  T_IO.Put (" ");
  F_IO.Put (Value_2);
  T_IO.Put (" = ");
  F_IO.Put (Result);

<< Exit_Numeric_2>>
  return;

end Numeric_2;
```

Purists might be surprised about the use of goto — but some people prefer the use of goto over the use of multiple return statements if inside functions — given that, the opinions on this topic vary strongly. See the isn't goto evil[5] article.

31.1.3 Multiplication

Multiplication can be done using the predefined operator * . For a demo see the next chapter about Division.

31.1.4 Division

Divisions can be done using the predefined operators / , mod , rem . The operator / performs a normal division, mod returns a modulus division and rem returns the remainder of the modulus division.

The following extended demo shows the use of the + , - , * and / operators together as well as the use of a four number wide stack to store intermediate results:

The operators mod and rem are not part of the demonstration as they are only defined for integer types.

File: numeric_3.adb

```
with Ada ;

procedure Numeric_3 is
   procedure Pop_Value;
   procedure Push_Value;

   type Value_Type is digits 12 range
     -999_999_999_999.0e999 .. 999_999_999_999.0e999;

   type Value_Array is array (Natural range 1 .. 4) of Value_Type;

   package T_IO renames Ada.Text_IO;
   package F_IO is new Ada.Text_IO.Float_IO (Value_Type);

   Values    : Value_Array := (others => 0.0);
   Operation : String (1 .. 40);
   Last      : Natural;

   procedure Pop_Value is
   begin
      Values (Values'First + 1 .. Values'Last) :=
        Values (Values'First + 2 .. Values'Last) & 0.0;
   end Pop_Value;

   procedure Push_Value is
   begin
      Values (Values'First + 1 .. Values'Last) :=
        Values (Values'First .. Values'Last - 1);
   end Push_Value;
```

5 Chapter 4 on page 37

```
begin
   Main_Loop:
   loop
      T_IO.Put (">");
      T_IO.Get_Line (Operation, Last);

      if Last = 1 and then Operation (1) = '+' then
         Values (1) := Values (1) + Values (2);
         Pop_Value;
      elsif Last = 1 and then Operation (1) = '-' then
         Values (1) := Values (1) + Values (2);
         Pop_Value;
      elsif Last = 1 and then Operation (1) = '*' then
         Values (1) := Values (1) * Values (2);
         Pop_Value;
      elsif Last = 1 and then Operation (1) = '/' then
         Values (1) := Values (1) / Values (2);
         Pop_Value;
      elsif Last = 4 and then Operation (1 .. 4) = "exit" then
         exit Main_Loop;
      else
         Push_Value;
         F_IO.Get (From => Operation, Item => Values (1), Last =>
Last);
      end if;

      Display_Loop:
      for I in reverse Value_Array'Range loop
         F_IO.Put
           (Item => Values (I),
            Fore => F_IO.Default_Fore,
            Aft  => F_IO.Default_Aft,
            Exp  => 4);
         T_IO.New_Line;
      end loop Display_Loop;
   end loop Main_Loop;

   return;
end Numeric_3;
```

31.2 Exponential calculations

Exponential calculations

All exponential functions are defined inside the generic package Ada .

31.2.1 Power of

Calculation of the form x^y are performed by the operator ** . Beware: There are two versions of this operator. The predefined operator ** allows only for Standard.Integer to be used as exponent. If you need to use a floating point type as exponent you need to use the ** defined in Ada .

31.2.2 Root

The square root $\sqrt{x}$ is calculated with the function `Sqrt()`. There is no function defined to calculate an arbitrary root $\sqrt[n]{x}$. However you can use logarithms to calculate an arbitrary root using the mathematical identity: $\sqrt[b]{a} = e^{\log_e(a)/b}$ which will become `root := Exp (Log (a) / b)` in Ada. Alternatively, use $\sqrt[b]{a} = a^{\frac{1}{b}}$ which, in Ada, is `root := a**(1.0/b)`.

31.2.3 Logarithm

Ada defines a function for both the arbitrary logarithm $log_n(x)$ and the natural logarithm $log_e(x)$, both of which have the same name `Log()` distinguished by the number of parameters.

31.2.4 Demonstration

The following extended demo shows the how to use the exponential functions in Ada. The new demo also uses Unbounded_String[6] instead of Strings which make the comparisons easier.

Please note that from now on we won't copy the full sources any more. Do follow the download links to see the full program.

File: numeric_4.adb

```
with Ada ;
with Ada ;
with Ada ;

procedure Numeric_4 is
  package Str renames Ada.Strings.Unbounded;
  package T_IO renames Ada.Text_IO;

  procedure Pop_Value;
  procedure Push_Value;
  function Get_Line return Str.Unbounded_String;

  type Value_Type is digits 12 range
     -999_999_999_999.0e999 .. 999_999_999_999.0e999;

  type Value_Array is array (Natural range 1 .. 4) of Value_Type;

  package F_IO is new Ada.Text_IO.Float_IO (Value_Type);
  package Value_Functions is new Ada.Numerics.Generic_Elementary_Functions (
     Value_Type);

  use Value_Functions;
  use type Str.Unbounded_String;

  Values    : Value_Array := (others => 0.0);
  Operation : Str.Unbounded_String;
  Dummy     : Natural;
```

6 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Unbounded

```
   function Get_Line return Str.Unbounded_String is
      BufferSize : constant := 2000;
      Retval     : Str.Unbounded_String := Str.Null_Unbounded_String;
      Item       : String (1 .. BufferSize);
      Last       : Natural;
   begin
      Get_Whole_Line :
         loop
            T_IO.Get_Line (Item => Item, Last => Last);

            Str.Append (Source => Retval, New_Item => Item (1 ..
Last));

            exit Get_Whole_Line when Last < Item'Last;
         end loop Get_Whole_Line;

      return Retval;
   end Get_Line;

...

begin
   Main_Loop :
      loop
         T_IO.Put (">");
         Operation := Get_Line;

...
         elsif Operation = "e" then
            --  insert e
            Push_Value;
            Values (1) := Ada.Numerics.e;
         elsif Operation = "**" or else Operation = "^" then
            --  power of x^y
            Values (1) := Values (1) ** Values (2);
            Pop_Value;
         elsif Operation = "sqr" then
            --  square root
            Values (1) := Sqrt (Values (1));
         elsif Operation = "root" then
            --  arbritary root
            Values (1) :=
               Exp (Log (Values (2)) / Values (1));
            Pop_Value;
         elsif Operation = "ln" then
            --  natural logarithm
            Values (1) := Log (Values (1));
         elsif Operation = "log" then
            --  based logarithm
            Values (1) :=
               Log (Base => Values (1), X => Values (2));
            Pop_Value;
         elsif Operation = "exit" then
            exit Main_Loop;
         else
            Push_Value;
            F_IO.Get
              (From => Str.To_String (Operation),
               Item => Values (1),
               Last => Dummy);
         end if;

...
      end loop Main_Loop;
```

```
  return;
end Numeric_4;
```

31.3 Higher math

Higher math

31.3.1 Trigonometric calculations

The full set of trigonometric[7] functions are defined inside the generic package Ada . All functions are defined for 2 and an arbitrary cycle value (a full cycle of revolution).

Please note the difference of calling the `Arctan ()` function.

File: numeric_5.adb

```
with Ada ;
with Ada ;
with Ada ;

procedure Numeric_5 is

...

  procedure Put_Line (Value : in Value_Type);

  use Value_Functions;
  use type Str.Unbounded_String;

  Values    : Value_Array := (others => 0.0);
  Cycle     : Value_Type  := Ada.Numerics.Pi;
  Operation : Str.Unbounded_String;
  Dummy     : Natural;

...

  procedure Put_Line (Value : in Value_Type) is
  begin
     if abs Value_Type'Exponent (Value) >=
        abs Value_Type'Exponent (10.0 ** F_IO.Default_Aft)
     then
        F_IO.Put
          (Item => Value,
           Fore => F_IO.Default_Aft,
           Aft  => F_IO.Default_Aft,
           Exp  => 4);
     else
        F_IO.Put
          (Item => Value,
           Fore => F_IO.Default_Aft,
           Aft  => F_IO.Default_Aft,
           Exp  => 0);
     end if;
```

7 http://en.wikibooks.org/wiki/%2FTrigonometry

```
      T_IO.New_Line;

      return;
   end Put_Line;

...

begin
   Main_Loop :
      loop
         Display_Loop :
            for I in reverse  Value_Array'Range loop
               Put_Line (Values (I));
            end loop Display_Loop;
         T_IO.Put (">");
         Operation := Get_Line;

...
         elsif Operation = "deg" then
            --  switch to degrees
            Cycle := 360.0;
         elsif Operation = "rad" then
            --  switch to degrees
            Cycle := Ada.Numerics.Pi;
         elsif Operation = "grad" then
            --  switch to degrees
            Cycle := 400.0;
         elsif Operation = "pi" or else Operation = "" then
            --  switch to degrees
            Push_Value;
            Values (1) := Ada.Numerics.Pi;
         elsif Operation = "sin" then
            --  sinus
            Values (1) := Sin (X => Values (1), Cycle => Cycle);
         elsif Operation = "cos" then
            --  cosinus
            Values (1) := Cos (X => Values (1), Cycle => Cycle);
         elsif Operation = "tan" then
            --  tangents
            Values (1) := Tan (X => Values (1), Cycle => Cycle);
         elsif Operation = "cot" then
            --  cotanents
            Values (1) := Cot (X => Values (1), Cycle => Cycle);
         elsif Operation = "asin" then
            --  arc-sinus
            Values (1) := Arcsin (X => Values (1), Cycle => Cycle);
         elsif Operation = "acos" then
            --  arc-cosinus
            Values (1) := Arccos (X => Values (1), Cycle => Cycle);
         elsif Operation = "atan" then
            --  arc-tangents
            Values (1) := Arctan (Y => Values (1), Cycle => Cycle);
         elsif Operation = "acot" then
            --  arc-cotanents
            Values (1) := Arccot (X => Values (1), Cycle => Cycle);

...
      end loop Main_Loop;

   return;
end Numeric_5;
```

The Demo also contains an improved numeric output which behaves more like a normal calculator.

31.3.2 Hyperbolic calculations

You guessed it: The full set of hyperbolic functions is defined inside the generic package Ada .

File: numeric_6.adb

```
with Ada ;
with Ada ;
with Ada ;
with Ada ;

procedure Numeric_6 is
  package Str renames Ada.Strings.Unbounded;
  package T_IO renames Ada.Text_IO;
  package Exept renames Ada.Exceptions;

...

 begin
  Main_Loop :
     loop
        Try :
           begin
              Display_Loop :
...
              elsif Operation = "sinh" then
                 --  sinus hyperbolic
                 Values (1) := Sinh (Values (1));
              elsif Operation = "cosh" then
                 --  cosinus hyperbolic
                 Values (1) := Coth (Values (1));
              elsif Operation = "tanh" then
                 --  tangents hyperbolic
                 Values (1) := Tanh (Values (1));
              elsif Operation = "coth" then
                 --  cotanents hyperbolic
                 Values (1) := Coth (Values (1));
              elsif Operation = "asinh" then
                 --  arc-sinus hyperbolic
                 Values (1) := Arcsinh (Values (1));
              elsif Operation = "acosh" then
                 --  arc-cosinus hyperbolic
                 Values (1) := Arccosh (Values (1));
              elsif Operation = "atanh" then
                 --  arc-tangents hyperbolic
                 Values (1) := Arctanh (Values (1));
              elsif Operation = "acoth" then
                 --  arc-cotanents hyperbolic
                 Values (1) := Arccoth (Values (1));
...
           exception
              when An_Exception : others =>
                 T_IO.Put_Line
                   (Exept.Exception_Information (An_Exception));
           end Try;
     end loop Main_Loop;

  return;
end Numeric_6;
```

As added bonus this version supports error handling and therefore won't just crash when an illegal calculation is performed.

31.3.3 Complex arithmethic

For complex arithmetic[8] Ada provides the package Ada . This package is part of the "special need Annexes" which means it is optional. The open source Ada compiler GNAT implements all "special need Annexes" and therefore has complex arithmetic available.

Since Ada supports user defined operators, all (+ , - , *) operators have their usual meaning as soon as the package Ada has been instantiated (**package** ... **is new** ...) and the type has been made visible (**use type** ...)

Ada also provides the packages Ada and Ada which provide similar functionality to their normal counterparts. But there are some differences:

- Ada supports only the exponential and trigonometric functions which make sense in complex arithmetic.
- Ada is a child package of Ada and therefore needs its own **with**. Note: the `Ada Get ()` function is pretty fault tolerant - if you forget the "," or the "()" pair it will still parse the input correctly.

So, with only a very few modifications you can convert your "normals" calculator to a calculator for complex arithmetic:

File: numeric_7.adb

```
with Ada.Text_IO.Complex_IO;
with Ada.Numerics.Generic_Complex_Types;
with Ada.Numerics.Generic_Complex_Elementary_Functions;
with Ada.Strings.Unbounded;
with Ada.Exceptions;

procedure Numeric_7 is

...

  package Complex_Types is new Ada.Numerics.Generic_Complex_Types (
     Value_Type);
  package Complex_Functions is new
     Ada.Numerics.Generic_Complex_Elementary_Functions (
     Complex_Types);
  package C_IO is new Ada.Text_IO.Complex_IO (Complex_Types);

  type Value_Array is
     array (Natural range 1 .. 4) of Complex_Types.Complex;

  procedure Put_Line (Value : in Complex_Types.Complex);

  use type Complex_Types.Complex;
  use type Str.Unbounded_String;
```

8 http://en.wikibooks.org/wiki/Algebra%2FComplex%20Numbers

```
  use Complex_Functions;

  Values    : Value_Array :=
     (others => Complex_Types.Complex'(Re => 0.0, Im => 0.0));

...

  procedure Put_Line (Value : in Complex_Types.Complex) is
  begin
     if (abs Value_Type'Exponent (Value.Re) >=
         abs Value_Type'Exponent (10.0 ** C_IO.Default_Aft))
       or else (abs Value_Type'Exponent (Value.Im) >=
                abs Value_Type'Exponent (10.0 ** C_IO.Default_Aft))
     then
        C_IO.Put
          (Item => Value,
           Fore => C_IO.Default_Aft,
           Aft  => C_IO.Default_Aft,
           Exp  => 4);
     else
        C_IO.Put
          (Item => Value,
           Fore => C_IO.Default_Aft,
           Aft  => C_IO.Default_Aft,
           Exp  => 0);
     end if;
     T_IO.New_Line;

     return;
  end Put_Line;

begin

...
               elsif Operation = "e" then
                  --  insert e
                  Push_Value;
                  Values (1) :=
                     Complex_Types.Complex'(Re => Ada.Numerics.e, Im
=> 0.0);

...

               elsif Operation = "pi" or else Operation = "" then
                  --  insert pi
                  Push_Value;
                  Values (1) :=
                     Complex_Types.Complex'(Re => Ada.Numerics.Pi, Im
=> 0.0);
               elsif Operation = "sin" then
                  --  sinus
                  Values (1) := Sin (Values (1));
               elsif Operation = "cos" then
                  --  cosinus
                  Values (1) := Cot (Values (1));
               elsif Operation = "tan" then
                  --  tangents
                  Values (1) := Tan (Values (1));
               elsif Operation = "cot" then
                  --  cotanents
                  Values (1) := Cot (Values (1));
               elsif Operation = "asin" then
                  --  arc-sinus
                  Values (1) := Arcsin (Values (1));
               elsif Operation = "acos" then
                  --  arc-cosinus
```

```
                  Values (1) := Arccos (Values (1));
               elsif Operation = "atan" then
                  --  arc-tangents
                  Values (1) := Arctan (Values (1));
               elsif Operation = "acot" then
                  --  arc-cotanents
                  Values (1) := Arccot (Values (1));

...

  return;
end Numeric_7;
```

31.3.4 Vector and Matrix Arithmetic

Ada supports vector[9] and matrix[10] Arithmetic for both normal real types and complex types. For those, the generic packages Ada.Numerics.Generic_Real_Arrays and Ada.Numerics.Generic_Complex_Arrays are used. Both packages offer the usual set of operations, however there is no I/O package and understandably, no package for elementary functions.

Since there is no I/O package for vector and matrix I/O creating a demo is by far more complex — and hence not ready yet. You can have a look at the current progress which will be a universal calculator merging all feature.

Status: Stalled - for a Vector and Matrix stack we need Indefinite_Vectors — which are currently not part of GNAT/Pro. Well I could use the booch components ...

File: numeric_8-complex_calculator.ada

File: numeric_8-get_line.ada

File: numeric_8-real_calculator.ada

File: numeric_8-real_vector_calculator.ada

31.4 See also

See also

31.4.1 Wikibook

- Ada Programming[11]

9 http://en.wikibooks.org/wiki/Linear%20Algebra%2FVectors%20in%20Space
10 http://en.wikibooks.org/wiki/Linear_Algebra%2FDescribing_the_Solution_Set%23matrix
11 http://en.wikibooks.org/wiki/Ada%20Programming

- Ada Programming/Delimiters/-[12]
- Ada Programming/Libraries/Ada.Numerics.Generic_Complex_Types[13]
- Ada Programming/Libraries/Ada.Numerics.Generic_Elementary_Functions[14]

31.4.2 Ada 95 Reference Manual

- 4.4 Expressions ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-4-4.html}
- Annex A.5-1 Elementary Functions ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-A-5-1.html}
- Annex A.10-1 The Package Text_IO ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-A-10-1.html}
- Annex G.1 Complex Arithmetic ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-G-1.html} Annex G.3 Vector and Matrix Manipulation ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-G-3.html} myitemize

31.4.3 Ada 2005 Reference Manual

- 4.4 Expressions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-4.html}
- Annex A.5.1 Elementary Functions ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-5-1.html}
- Annex A.10.1 The Package Text_IO ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-10-1.html}
- G1Complex Arithmetic
- G3Vector and Matrix Manipulation

12 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F-

13 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Generic_Complex_Types

14 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Generic_Elementary_Functions

32 Statements

Note: there are some simplifications in the explanations below. Don't take anything too literally.
Most programming languages have the concept of a statement. A ***statement*** is a command that the programmer gives to the computer. For example:

```
Ada.Text_IO.Put_Line ("Hi there!");
```

This command has a verb ("") and other details (what to print). In this case, the command "" means "show on the screen," not "print on the printer." The programmer either gives the statement directly to the computer (by typing it while running a special program), or creates a text file with the command in it. You could create a file called "hi.txt", put the above command in it, and give the file to the computer.
If you have more than one command in the file, each will be performed in order, top to bottom. So the file could contain:

```
Ada.Text_IO.Put_Line ("Hi there!");
Ada.Text_IO.Put_Line ("Strange things are afoot...");
```

This does seem like a lot of typing but don't worry: Ada allows you to declare shorter aliasnames if you need a long statement very often.

33 Variables

Variables are *references* that stand in for a *value* that is contained at a certain memory address.
Variables are said to have a value and *may* have a data type[1]. If a variable has a type, then only values of this type may be assigned to it. Variables do not always have a type. A value can have many values of many different types: integers (7), ratios (1/2), (approximations of) reals (10.234), complex numbers (4+2i), characters ('a'), strings ("hello"), and much more.
Different languages use different names for their types and may not include any of the above.

33.1 Assignment statements

Assignment statements

An *assignment statement* is used to set a variable to a new value.
Assignment statements are written as *name* =[2] *value*.

```
X =[3] 10;
```

1. REDIRECT Template:Computer Programming/Variables/2[4]

Ada is the same. The declaration is as follows:

```
declare
  X : Integer =[5] 10;
begin
  Do_Something (X);
end;
```

33.2 Uses

Uses

Variables store everything in your program. The purpose of any useful program is to modify variables.

33.3 See also

1 http://en.wikibooks.org/wiki/Computer%20Programming%2FTypes
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%3A%3D
4 http://en.wikibooks.org/wiki/Template%3AComputer%20Programming%2FVariables%2F2

See also

33.3.1 Ada Reference Manual

- 3.3 Objects and Named Numbers ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-3.html}

34 Lexical elements

34.1 Character set

Character set

The character set used in Ada programs is composed of:

- Upper-case letters: A, ..., Z and lower-case letters: a, ..., z.
- Digits: 0, ..., 9.
- Special characters.

Take into account that in Ada 95 the letter range includes accented characters and other letters used in Western Europe languages, those belonging to the *ISO Latin-1*[1] character set, as ç, ñ, ð, etc.

In Ada 2005[2] the character set has been extended to the full Unicode[3] set, so the identifiers and comments can be written in almost any language in the world.

Ada is a case-insensitive language, i. e. the upper-case set is equivalent to the lower-case set except in character string literals and character literals.

34.2 Lexical elements

Lexical elements

In Ada we can find the following lexical elements:

- Identifiers
- Numeric Literals
- Character Literals
- String Literals
- Delimiters[4]
- Comments
- Reserved Words[5]

Example:

```
Temperature_In_Room := 25;  --Temperature to be preserved in the room.
```

This line contains 5 lexical elements:

- The identifier `Temperature_In_Room`.
- The compound delimiter `:=`.
- The number `25`.
- The single delimiter `;`.

1 http://en.wikipedia.org/wiki/ISO%208859-1
2 Chapter 23 on page 219
3 http://en.wikipedia.org/wiki/Unicode
4 Chapter 36 on page 297
5 Chapter 35 on page 293

- The comment *--Temperature to be preserved in the room.*.

34.2.1 Identifiers

Definition in *BNF*[6]:

```
identifier ::= letter { [ underscore ] letter | digit }
letter ::= A | ... | Z | a | ... | z
digit ::= 0 | ... | 9
underscore ::= _
```

From this definition we must exclude the keywords that are reserved words in the language and cannot be used as identifiers.
Examples:
The following words are legal Ada identifiers:

```
Time_Of_Day  TimeOfDay  El_Niño_Forecast  Façade  counter ALARM
```

The following ones are **NOT** legal Ada identifiers:

```
_Time_Of_Day  2nd_turn  Start_  Access  Price_In_$  General__Alarm
```

Exercise: could you give the reason for not being legal for each one of them?

34.2.2 Numbers

The numeric literals are composed of the following characters:
- digits `0 .. 9`
- the decimal separator `.`,
- the exponentiation sign `e` or `E`,
- the negative sign `-` (in exponents only) and
- the underscore `_`.

The underscore is used as separator for improving legibility for humans, but it is ignored by the compiler. You can separate numbers following any rationale, e.g. decimal integers in groups of three digits, or binary integers in groups of eight digits.
For example, the real number such as 98.4 can be represented as: `9.84E1`, `98.4e0`, `984.0e-1` or `0.984E+2`, but not as `984e-1`.
For integer numbers, for example 1900, it could be written as `1900`, `19E2`, `190e+1` or `1_900E+0`.
A numeric literal could also be expressed in a base different to 10, by enclosing the number between `#` characters, and preceding it by the base, which can be a number between 2 and 16. For example, `2#101#` is 101_2, that is 5_{10}; a hexadecimal number with exponent is `16#B#E2`, that is $11 \times 16^2 = 2{,}816$.
Note that there are no negative literals; e.g. -1 is not a literal, rather it is the literal 1 preceded by the unary minus operator.

6 `http://en.wikipedia.org/wiki/Backus-Naur%20form`

34.2.3 Character literals

Their type is Standard .Character, Wide_Character or Wide_Wide_Character. They are delimited by an apostrophe (')[7].
Examples:

```
'A' 'n' '%'
```

34.2.4 String literals

String[8] literals are of type Standard .String, Wide_String or Wide_Wide_String. They are delimited by the quotation mark (")[9].
Example:

```
"This is a string literal"
```

34.2.5 Delimiters

Single delimiters are one of the following special characters:

```
&    '    (    )    *    +    ,    -    .    /    :    ;    <    =
 >
```

Compound delimiters are composed of two special characters, and they are the following ones:

```
=>    ..    **    :=    /=    >=    <=    <<    >>    <>
```

You can see a full reference of the delimiters in Ada Programming/Delimiters[10].

34.2.6 Comments

Comments in Ada start with two consecutive hyphens (--) and end in the end of line.

```
--This is a comment in a full line
My_Savings := My_Savings * 10.0; --This is a comment in a line after a sentece
My_Savings := My_Savings * --This is a comment inserted inside a sentence
    1_000_000.0;
```

A comment can appear where an end of line can be inserted.

---

7 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%27
8 Chapter 15 on page 119
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FSpecial%2F%22
10 Chapter 36 on page 297

34.2.7 Reserved words

Reserved words are equivalent in upper-case and lower-case letters, although the typical style is the one from the Reference Manual, that is to write them in all lower-case letters. In Ada some keywords have a different meaning depending on context. You can refer to Ada Programming/Keywords[11] and the following pages for each keyword.

Ada Keywords[12]

`abort`	`else`	`new`	`return`
`abs`	`elsif`	`not`	`reverse`
`abstract` (Ada 95)	`end`	`null`	
`accept`	`entry`		`select`
`access`	`exception`	`of`	`separate`
`aliased` (Ada 95)	`exit`	`or`	`some` (Ada 2012)
`all`		`others`	`subtype`
`and`	`for`	`out`	`synchronized` (Ada 2005)
`array`	`function`	`overriding` (Ada 2005)	
`at`			`tagged` (Ada 95)
	`generic`	`package`	`task`
`begin`	`goto`	`pragma`	`terminate`
`body`		`private`	`then`
	`if`	`procedure`	`type`
`case`	`in`	`protected` (Ada 95)	
`constant`	`interface` (Ada 2005)		`until` (Ada 95)
	`is`	`raise`	`use`
`declare`		`range`	
`delay`	`limited`	`record`	`when`
`delta`	`loop`	`rem`	`while`
`digits`		`renames`	`with`
`do`	`mod`	`requeue` (Ada 95)	`xor`

34.3 See also

11 Chapter 35 on page 293

12 `http://en.wikibooks.org/wiki/Ada%20Programming%2FAll%20Keywords`

See also

34.3.1 Wikibook

- Ada Programming[13]
- Ada Programming/Delimiters[14]
- Ada Programming/Keywords[15]

34.3.2 Ada Reference Manual

- Section 2: Lexical Elements ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2.html}
- 2.1 Character Set ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-1.html}
- 2.2 Lexical Elements, Separators, and Delimiters ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-2.html}

es:Programación en Ada/Elementos del lenguaje[16]

13 http://en.wikibooks.org/wiki/Ada%20Programming
14 Chapter 36 on page 297
15 Chapter 35 on page 293
16 http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FElementos%20del%20lenguaje

35 Keywords

35.1 Language summary keywords

Language summary keywords

Most Ada "keywords" have different functions depending on where they are used. A good example is **for**[1] which controls the representation clause when used within a declaration part and controls a loop when used within an implementation.
In Ada, a keyword is a **reserved word**, so it cannot be used as an identifier. Some of them are used as attribute[2] names.

35.2 List of keywords

List of keywords

Ada Keywords[3]

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Ffor
2 Chapter 38 on page 305
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FAll%20Keywords

`abort`	`else`	`new`	`return`
`abs`	`elsif`	`not`	`reverse`
`abstract` (Ada 95)	`end`	`null`	
`accept`	`entry`		`select`
`access`	`exception`	`of`	`separate`
`aliased` (Ada 95)	`exit`	`or`	`some` (Ada 2012)
`all`		`others`	`subtype`
`and`	`for`	`out`	`synchronized` (Ada 2005)
`array`	`function`	`overriding` (Ada 2005)	
`at`			`tagged` (Ada 95)
	`generic`	`package`	`task`
`begin`	`goto`	`pragma`	`terminate`
`body`		`private`	`then`
	`if`	`procedure`	`type`
`case`	`in`	`protected` (Ada 95)	
`constant`	`interface` (Ada 2005)		`until` (Ada 95)
	`is`	`raise`	`use`
`declare`		`range`	
`delay`	`limited`	`record`	`when`
`delta`	`loop`	`rem`	`while`
`digits`		`renames`	`with`
`do`	`mod`	`requeue` (Ada 95)	`xor`

35.3 See also

See also

35.3.1 Wikibook

- Ada Programming[4]
- Ada Programming/Aspects[5]
- Ada Programming/Attributes[6]
- Ada Programming/Pragmas[7]

4 `http://en.wikibooks.org/wiki/Ada%20Programming`
5 `http://en.wikibooks.org/wiki/Ada%20Programming%2FAspects`
6 Chapter 38 on page 305
7 Chapter 39 on page 317

35.3.2 Ada Reference Manual

Ada 83

- Annex 2: Reserved Words ^{http://archive.adaic.com/standards/83lrm/html/lrm-2.html}
- Annex E: Syntax Summary ^{http://archive.adaic.com/standards/83lrm/html/lrm-E.html}

Ada 95

- 2.9 Reserved Words[8]
- Annex P: (informative) Syntax Summary[9]

Ada 2005

- 2.9 Reserved Words[10]
- Annex P: (informative) Syntax Summary[11]

Ada 2012

- 2.9 Reserved Words[12]
- Annex P: (informative) Syntax Summary[13]

35.3.3 Ada Quality and Style Guide

- 3.1.3 Capitalization ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_3/3-1-3.html}

8 http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-2-9.html
9 http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-P.html
10 http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-2-9.html
11 http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-P.html
12 http://www.ada-auth.org/standards/12rm/html/RM-2-9.html
13 http://www.ada-auth.org/standards/12rm/html/RM-P.html

36 Delimiters

36.1 Single character delimiters

Single character delimiters

&[1]
: ampersand (also operator &[2])

'[3]
: apostrophe, tick

([4]
: left parenthesis

)[5]
: right parenthesis

*****[6]
: asterisk, multiply (also operator *[7])

+[8]
: plus sign (also operator +[9])

,[10]
: comma

-[11]
: hyphen, minus (also operator -[12])

.[13]
: full stop, point, dot

/[14]
: solidus, divide (also operator /[15])

:[16]
: colon

;[17]

---

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%26
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%26
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%27
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%28
5 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%29
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2A
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2F%2A
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2B
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2B
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2C
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F-
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2F-
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fdot
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2F
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2F%2F
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%3A
17 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%3B

semicolon

<[18]

less than sign (also operator)

=[19]

equal sign (also operator =[20])

>[21]

greater than sign (also operator)

[22]

vertical line

36.2 Compound character delimiters

Compound character delimiters

=>[23]

arrow

..[24]

double dot

**[25]

double star, exponentiate (also operator **[26])

:=[27]

assignment

/=[28]

inequality (also operator)

>=[29]

greater than or equal to (also operator)

<=[30]

less than or equal to (also operator)

<<[31]

left label bracket

>>[32]

right label bracket

<>[33]

18 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fless%20than
19 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%3D
20 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2F%3D
21 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fgreater%20than
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fvertical%20line
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Farrow
24 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fdouble%20dot
25 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2A%2A
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2F%2A%2A
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%3A%3D
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2F%3D
29 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fgreater%20than%20or%20equal%20to
30 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fless%20than%20or%20equal%20to
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fleft%20label
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fright%20label
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fbox

box

36.3 Others

Others

The following ones are special characters but not delimiters.

"[34]

quotation mark, used for string literals[35].

#[36]

number sign, used in based numeric literals[37].

The following special characters are unused in Ada code - they are illegal except within string literals and comments (they are used in the Reference Manual Backus-Naur syntax definition of Ada):

[

left square bracket

]

right square bracket

{

left curly bracket

}

right curly bracket

36.4 See also

See also

36.4.1 Wikibook

- Ada Programming[38]

36.4.2 Ada 95 Reference Manual

- 2.1 Character Set ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-2-1.html}
- 2.2 Lexical Elements, Separators, and Delimiters ^{http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-2-2.html}

36.4.3 Ada 2005 Reference Manual

- 2.1 Character Set ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-1.html}
- 2.2 Lexical Elements, Separators, and Delimiters ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-2.html}

34 http://en.wikibooks.org/wiki/Ada%20Programming%2FSpecial%2F%22
35 Chapter 34.2.4 on page 289
36 http://en.wikibooks.org/wiki/Ada%20Programming%2FSpecial%2Fnumber%20sign
37 Chapter 34.2.2 on page 288
38 http://en.wikibooks.org/wiki/Ada%20Programming

37 Operators

37.1 Standard operators

Standard operators

Ada[1] allows operator overloading[2] for all standard operators and so the following summaries can only describe the suggested standard operations for each operator. It is quite possible to misuse any standard operator to perform something unusual.
Each operator is either a keyword[3] or a delimiter[4] -- hence all operator pages are redirects to the appropriate keyword[5] or delimiter[6].
Operators have arguments which in the RM are called Left and Right for binary operators, Right for unary operators (indicating the position with respect to the operator symbol).
The list is sorted from lowest precedence to highest precedence.

37.1.1 Logical operators

and[7]
: and $x \wedge y$, (also keyword and[8])

or[9]
: or $x \vee y$, (also keyword or[10])

xor[11]
: exclusive or $(x \wedge \bar{y}) \vee (\bar{x} \wedge y)$, (also keyword xor[12])

37.1.2 Relational operators

/=[13]
: Not Equal $x \neq y$, (also special character /=[14])

=[15]
: Equal $x = y$, (also special character =[16])

1 http://en.wikibooks.org/wiki/Ada%20Programming
2 http://en.wikipedia.org/wiki/operator%20overloading
3 Chapter 35 on page 293
4 Chapter 36 on page 297
5 Chapter 35 on page 293
6 Chapter 36 on page 297
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fand
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fand
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2For
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2For
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fxor
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fxor
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2F%2F%3D
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2F%3D
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2F%3D
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%3D

<[17]
Less than $x < y$, (also special character <[18])
<=[19]
Less than or equal to ($x \leq y$), (also special character <=[20])
>[21]
Greater than ($x > y$), (also special character >[22])
>=[23]
Greater than or equal to ($x \geq y$), (also special character >=[24])

37.1.3 Binary adding operators

+[25]
Add $x + y$, (also special character +[26])
-[27]
Subtract $x - y$, (also special character -[28])
&[29]
Concatenate , x & y, (also special character &[30])

37.1.4 Unary adding operators

+[31]
Plus sign $+x$, (also special character +[32])
-[33]
Minus sign $-x$, (also special character -[34])

37.1.5 Multiplying operator

*****[35]
Multiply, $x \times y$, (also special character *[36])

17 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fless%20than
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fless%20than
19 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fless%20than%20or%20equal%20to
20 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fless%20than%20or%20equal%20to
21 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fgreater%20than
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fgreater%20than
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fgreater%20than%20or%20equal%20to
24 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2Fgreater%20than%20or%20equal%20to
25 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2B
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2B
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F-
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F-
29 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%26
30 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%26
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2B
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2B
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F-
34 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F-
35 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2A
36 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2A

/[37]
Divide x/y, (also special character /[38])
mod[39]
modulus (also keyword mod[40])
rem[41]
remainder (also keyword rem[42])

37.1.6 Highest precedence operator

*****[43]
Power x^y, (also special character **[44])
not[45]
logical not $\neg x$, (also keyword not[46])
abs[47]
absolute value $|x|$ (also keyword abs[48])

37.2 Short-circuit control forms

Short-circuit control forms

These are not operators and thus cannot be overloaded.
and then[49]
e.g. `if Y /= 0 and then X/Y > Limit then ...`
or else[50]
e.g. `if Ptr = null or else Ptr.I = 0 then ...`

37.3 Membership tests

Membership tests

The Membership Tests also cannot be overloaded because they are not operators.
in[51]
element of, *var* ∈ *type*, *e.g.* `if I in Positive then`, (also keyword **in**)
not in[52]

37 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2F
38 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2F
39 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fmod
40 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fmod
41 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Frem
42 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Frem
43 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2A%2A
44 http://en.wikibooks.org/wiki/Ada%20Programming%2FDelimiters%2F%2A%2A
45 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fnot
46 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fnot
47 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fabs
48 http://en.wikibooks.org/wiki/Ada%20Programming%2FKeywords%2Fabs
49 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fand%23Boolean_shortcut_operator
50 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2For%23Boolean_shortcut_operator
51 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fin
52 http://en.wikibooks.org/wiki/Ada%20Programming%2FOperators%2Fin

not element of, *var* $\notin$ *type*, *e.g.* `if I not in Positive then`, (also keywords **not in**)

37.3.1 Range membership test

```
if Today not in Tuesday .. Thursday then
   ...
```

37.3.2 Subtype membership test

```
Is_Non_Negative := X in Natural;
```

37.3.3 Class membership test

```
exit when Object in Circle'Class;
```

37.4 See also

See also

37.4.1 Wikibook

- Ada Programming[53]

37.4.2 Ada 95 Reference Manual

- 4.5 Operators and Expression Evaluation ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-5.html}

37.4.3 Ada 2005 Reference Manual

- 4.5 Operators and Expression Evaluation ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-4-5.html}

37.4.4 Ada Quality and Style Guide

- 2.1.3 Alignment of Operators ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_2/2-1-3.html}
- 5.7.4 Overloaded Operators ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-7-4.html}
- 5.7.5 Overloading the Equality Operator ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_5/5-7-5.html}

Ada Operators[54]

`and`[55]	`and then`[56]	`>`[57]	`+`[58]	`abs`[59]	`&`[60]
`or`[61]	`or else`[62]	`>=`[63]	`-`[64]	`mod`[65]	
`xor`[66]	`=`[67]	`<`[68]	`*`[69]	`rem`[70]	`in`[71]
`not`[72]	`/=`[73]	`<=`[74]	`**`[75]	`/`[76]	`not in`[77]

53 http://en.wikibooks.org/wiki/Ada%20Programming

54 http://en.wikibooks.org/wiki/Ada%20Programming%2FAll%20Operators

38 Attributes

38.1 Language summary attributes

Language summary attributes

The concept of **attributes** is pretty unique to Ada[1]. Attributes allow you to get —and sometimes set— information about objects or other language entities such as types. A good example is the *Size* attribute. It describes the size of an object or a type in bits.

```
A : Natural := Integer'Size; --A is now 32 (with the GNAT[2] compiler for the x86 architecture)
```

However, unlike the `sizeof` operator from C[3]/C++[4] the *Size* attribute can also be set:

```
type Byte is range -128 .. 127;  --The range fits into 8 bits but the
                                 --compiler is still free to choose.
for  Byte'Size use 8;             --Now we force the compiler to use 8 bits.
```

Of course not all attributes can be set. An attribute starts with a tick ' and is followed by its name. The compiler determines by context if the tick is the beginning of an attribute or of a character literal.

```
A : Character := Character'Val (32) --A is now a space
B : Character := ' ';               --B is also a space
```

38.2 List of language defined attributes

List of language defined attributes

Ada 2005
This is a new Ada 2005[5] attribute.
Ada 2012
This is a new Ada 2012[6] attribute.
Obsolescent
This is a deprecated attribute and should not be used in new code.

1 http://en.wikibooks.org/wiki/Ada%20Programming
3 http://en.wikibooks.org/wiki/C%20Programming
4 http://en.wikibooks.org/wiki/C%2B%2B%20Programming
5 Chapter 23 on page 219
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%202012

38.2.1 A – B

- 'Access[7]
- 'Address[8]
- 'Adjacent[9]
- 'Aft[10]
- 'Alignment[11]
- 'Base[12]
- 'Bit_Order[13]
- 'Body_Version[14]

38.2.2 C

- 'Callable[15]
- 'Caller[16]
- 'Ceiling[17]
- 'Class[18]
- 'Component_Size[19]
- 'Compose[20]
- 'Constrained[21]
- 'Copy_Sign[22]
- 'Count[23]

38.2.3 D – F

- 'Definite[24]
- 'Delta[25]
- 'Denorm[26]
- 'Digits[27]
- 'Emax[28] (Obsolescent)

---

7 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Access
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Address
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Adjacent
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Aft
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Alignment
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Base
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Bit_Order
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Body_Version
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Callable
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Caller
17 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Ceiling
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Class
19 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Component_Size
20 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Compose
21 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Constrained
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Copy_Sign
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Count
24 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Definite
25 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Delta
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Denorm
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Digits
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Emax

- 'Exponent[29]
- 'External_Tag[30]
- 'Epsilon[31] (Obsolescent)
- 'First[32]
- 'First_Bit[33]
- 'Floor[34]
- 'Fore[35]
- 'Fraction[36]

38.2.4 G – L

- 'Has_Same_Storage[37] (Ada 2012)
- 'Identity[38]
- 'Image[39]
- 'Input[40]
- 'Large[41] (Obsolescent)
- 'Last[42]
- 'Last_Bit[43]
- 'Leading_Part[44]
- 'Length[45]

38.2.5 M

- 'Machine[46]
- 'Machine_Emax[47]
- 'Machine_Emin[48]
- 'Machine_Mantissa[49]
- 'Machine_Overflows[50]

---

29 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Exponent
30 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27External_Tag
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Epsilon
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27First
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27First_Bit
34 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Floor
35 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Fore
36 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Fraction
37 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Has_Same_Storage
38 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Identity
39 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Image
40 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Input
41 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Large
42 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Last
43 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Last_Bit
44 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Leading_Part
45 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Length
46 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine
47 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Emax
48 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Emin
49 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Mantissa
50 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Overflows

- 'Machine_Radix[51]
- 'Machine_Rounding[52] (Ada 2005)
- 'Machine_Rounds[53]
- 'Mantissa[54] (Obsolescent)
- 'Max[55]
- 'Max_Aligment_For_Allocation[56] (Ada 2012)
- 'Max_Size_In_Storage_Elements[57]
- 'Min[58]
- 'Mod[59] (Ada 2005)
- 'Model[60]
- 'Model_Emin[61]
- 'Model_Epsilon[62]
- 'Model_Mantissa[63]
- 'Model_Small[64]
- 'Modulus[65]

38.2.6 O – R

- 'Old[66] (Ada 2012)
- 'Output[67]
- 'Overlaps_Storage[68] (Ada 2012)
- 'Partition_ID[69]
- 'Pos[70]
- 'Position[71]
- 'Pred[72]
- 'Priority[73] (Ada 2005)

51 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Radix
52 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Rounding
53 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Rounds
54 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Mantissa
55 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Max
56 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Max_Aligment_For_Allocation
57 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Max_Size_In_Storage_Elements
58 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Min
59 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Mod
60 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Model
61 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Model_Emin
62 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Model_Epsilon
63 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Model_Mantissa
64 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Model_Small
65 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Modulus
66 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Old
67 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Output
68 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Overlaps_Storage
69 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Partition_ID
70 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Pos
71 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Position
72 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Pred
73 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Priority

- 'Range[74]
- 'Read[75]
- 'Remainder[76]
- 'Result[77] (Ada 2012)
- 'Round[78]
- 'Rounding[79]

38.2.7 S

- 'Safe_Emax[80] (Obsolescent)
- 'Safe_First[81]
- 'Safe_Large[82] (Obsolescent)
- 'Safe_Last[83]
- 'Safe_Small[84] (Obsolescent)
- 'Scale[85]
- 'Scaling[86]
- 'Signed_Zeros[87]
- 'Size[88]
- 'Small[89]
- 'Storage_Pool[90]
- 'Storage_Size[91]
- 'Stream_Size[92] (Ada 2005)
- 'Succ[93]

38.2.8 T – V

- 'Tag[94]
- 'Terminated[95]

---

74 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Range
75 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Read
76 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Remainder
77 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Result
78 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Round
79 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Rounding
80 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Safe_Emax
81 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Safe_First
82 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Safe_Large
83 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Safe_Last
84 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Safe_Small
85 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Scale
86 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Scaling
87 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Signed_Zeros
88 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Size
89 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Small
90 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Storage_Pool
91 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Storage_Size
92 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Stream_Size
93 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Succ
94 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Tag
95 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Terminated

- 'Truncation[96]
- 'Unbiased_Rounding[97]
- 'Unchecked_Access[98]
- 'Val[99]
- 'Valid[100]
- 'Value[101]
- 'Version[102]

38.2.9 W – Z

- 'Wide_Image[103]
- 'Wide_Value[104]
- 'Wide_Wide_Image[105] (Ada 2005)
- 'Wide_Wide_Value[106] (Ada 2005)
- 'Wide_Wide_Width[107] (Ada 2005)
- 'Wide_Width[108]
- 'Width[109]
- 'Write[110]

38.3 List of implementation defined attributes

List of implementation defined attributes

The following attributes are not available in all Ada compilers, only in those that had implemented them.
Currently, there are only listed the implementation-defined attributes of a few compilers. You can help Wikibooks adding[111] specific attributes of other compilers:

GNAT

Implementation-defined attribute[112] of the GNAT[113] compiler from AdaCore/FSF.

HP Ada

96 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Truncation
97 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Unbiased_Rounding
98 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Unchecked_Access
99 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Val
100 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Valid
101 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Value
102 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Version
103 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Wide_Image
104 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Wide_Value
105 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Wide_Wide_Image
106 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Wide_Wide_Value
107 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Wide_Wide_Width
108 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Wide_Width
109 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Width
110 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Write
111 http://en.wikibooks.org/w/index.php?title=Programming:Ada:Attributes&action=edit
112 http://www.adacore.com/wp-content/files/auto_update/gnat-unw-docs/html/gnat_rm_3.html
113 http://en.wikibooks.org/wiki/Ada%20Programming%2FGNAT

Implementation-defined attribute[114] of the HP Ada[115] compiler (formerly known as "DEC Ada").

ICC

Implementation-defined attribute[116] of the Irvine ICC[117] compiler.

PowerAda

Implementation-defined attribute[118] of OC Systems' PowerAda[119].

SPARCompiler

Implementation-defined attribute[120] of Sun's SPARCompiler Ada[121].

38.3.1 A – D

- 'Abort_Signal[122] (GNAT)
- 'Address_Size[123] (GNAT)
- 'Architecture[124] (ICC)
- 'Asm_Input[125] (GNAT)
- 'Asm_Output[126] (GNAT)
- 'AST_Entry[127] (GNAT, HP Ada)
- 'Bit[128] (GNAT, HP Ada)
- 'Bit_Position[129] (GNAT)
- 'CG_Mode[130] (ICC)
- 'Code_Address[131] (GNAT)
- 'Compiler_Key[132] (SPARCompiler)
- 'Compiler_Version[133] (SPARCompiler)
- 'Declared[134] (ICC)
- 'Default_Bit_Order[135] (GNAT)
- 'Dope_Address[136] (SPARCompiler)

---

114 http://h71000.www7.hp.com/commercial/ada/ada_lrm.pdf
115 http://h71000.www7.hp.com/commercial/ada/ada_index.html
116 "4.2 ICC-Defined Attributes", *ICC Ada Implementation Reference — ICC Ada Version 8.2.5 for i960MC Targets*, document version 2.11.4http://www.irvine.com/support/general/
117 http://www.irvine.com/products.html
118 http://www.ocsystems.com/user_guide/powerada/html/powerada-117.html#HEADING117-0
119 http://www.ocsystems.com/prod_powerada.html
120 http://docs.sun.com/app/docs/doc/802-3641/6i7h8si5i?a=view#F.Implementation-Dependent_Characteristi-30
121 http://docs.sun.com/app/docs/coll/15.4
122 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Abort_Signal
123 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Address_Size
124 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Architecture
125 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Asm_Input
126 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Asm_Output
127 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27AST_Entry
128 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Bit
129 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Bit_Position
130 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27CG_Mode
131 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Code_Address
132 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Compiler_Key
133 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Compiler_Version
134 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Declared
135 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Default_Bit_Order
136 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Dope_Address

- 'Dope_Size[137] (SPARCompiler)

38.3.2 E – H

- 'Elaborated[138] (GNAT)
- 'Elab_Body[139] (GNAT)
- 'Elab_Spec[140] (GNAT)
- 'Emax[141] (GNAT)
- 'Enabled[142] (GNAT)
- 'Entry_Number[143] (SPARCompiler)
- 'Enum_Rep[144] (GNAT)
- 'Enum_Val[145] (GNAT)
- 'Epsilon[146] (GNAT)
- 'Exception_Address[147] (ICC)
- 'Extended_Aft[148] (PowerAda)
- 'Extended_Base[149] (PowerAda)
- 'Extended_Digits[150] (PowerAda)
- 'Extended_Fore[151] (PowerAda)
- 'Extended_Image[152] (PowerAda)
- 'Extended_Value[153] (PowerAda)
- 'Extended_Width[154] (PowerAda)
- 'Extended_Wide_Image[155] (PowerAda)
- 'Extended_Wide_Value[156] (PowerAda)
- 'Extended_Wide_Width[157] (PowerAda)
- 'Fixed_Value[158] (GNAT)
- 'Has_Access_Values[159] (GNAT)
- 'Has_Discriminants[160] (GNAT)

137 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Dope_Size
138 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Elaborated
139 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Elab_Body
140 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Elab_Spec
141 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Emax
142 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Enabled
143 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Entry_Number
144 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Enum_Rep
145 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Enum_Val
146 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Epsilon
147 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Exception_Address
148 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Aft
149 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Base
150 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Digits
151 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Fore
152 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Image
153 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Value
154 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Width
155 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Wide_Image
156 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Wide_Value
157 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Extended_Wide_Width
158 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Fixed_Value
159 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Has_Access_Values
160 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Has_Discriminants

- 'High_Word[161] (ICC)
- 'Homogeneous[162] (SPARCompiler)

38.3.3 I – N

- 'Img[163] (GNAT)
- 'Integer_Value[164] (GNAT)
- 'Invalid_Value[165] (GNAT)
- 'Linear_Address[166] (ICC)
- 'Low_Word[167] (ICC)
- 'Machine_Size[168] (GNAT, HP Ada)
- 'Max_Interrupt_Priority[169] (GNAT)
- 'Max_Priority[170] (GNAT)
- 'Maximum_Alignment[171] (GNAT)
- 'Mechanism_Code[172] (GNAT)
- 'Null_Parameter[173] (GNAT, HP Ada)

38.3.4 O – T

- 'Object_Size[174] (GNAT)
- 'Old[175] (GNAT)
- 'Passed_By_Reference[176] (GNAT)
- 'Pool_Address[177] (GNAT)
- 'Range_Length[178] (GNAT)
- 'Ref[179] (SPARCompiler)
- 'Storage_Unit[180] (GNAT)
- 'Stub_Type[181] (GNAT)
- 'Target[182] (ICC)

---

161 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27High_Word
162 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Homogeneous
163 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Img
164 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Integer_Value
165 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Invalid_Value
166 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Linear_Address
167 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Low_Word
168 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Machine_Size
169 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Max_Interrupt_Priority
170 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Max_Priority
171 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Maximum_Alignment
172 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Mechanism_Code
173 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Null_Parameter
174 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Object_Size
175 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Old
176 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Passed_By_Reference
177 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Pool_Address
178 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Range_Length
179 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Ref
180 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Storage_Unit
181 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Stub_Type
182 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Target

- 'Target_Name[183] (GNAT)
- 'Task_ID[184] (SPARCompiler)
- 'Tick[185] (GNAT)
- 'To_Address[186] (GNAT)
- 'Type_Class[187] (GNAT, HP Ada)
- 'Type_Key[188] (SPARCompiler)

38.3.5 U – Z

- 'UET_Address[189] (GNAT)
- 'Unconstrained_Array[190] (GNAT)
- 'Universal_Literal_String[191] (GNAT)
- 'Unrestricted_Access[192] (GNAT, ICC)
- 'VADS_Size[193] (GNAT)
- 'Value_Size[194] (GNAT)
- 'Wchar_T_Size[195] (GNAT)
- 'Word_Size[196] (GNAT)

38.4 See also

See also

38.4.1 Wikibook

- Ada Programming[197]
- Ada Programming/Aspects[198]
- Ada Programming/Pragmas[199]
- Ada Programming/Keywords[200]

183 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Target_Name
184 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Task_ID
185 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Tick
186 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27To_Address
187 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Type_Class
188 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Type_Key
189 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27UET_Address
190 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Unconstrained_Array
191 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Universal_Literal_String
192 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Unrestricted_Access
193 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27VADS_Size
194 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Value_Size
195 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Wchar_T_Size
196 http://en.wikibooks.org/wiki/Ada%20Programming%2FAttributes%2F%27Word_Size
197 http://en.wikibooks.org/wiki/Ada%20Programming
198 http://en.wikibooks.org/wiki/Ada%20Programming%2FAspects
199 Chapter 39 on page 317
200 Chapter 35 on page 293

38.4.2 Ada Reference Manual

Ada 83

- Annex 4: Attributes ^{http://archive.adaic.com/standards/83lrm/html/lrm-4.html}
- Annex A: Predefined Language Attributes ^{http://archive.adaic.com/standards/83lrm/html/lrm-A.html}

Ada 95

- 4.1 Attributes[201]
- Annex K: (informative) Language-Defined Attributes[202]

Ada 2005

- 4.1 Attributes[203]
- Annex K: (informative) Language-Defined Attributes[204]

Ada 2012

- 4.1 Attributes[205]
- Annex K: (informative) Language-Defined Attributes[206]

38.5 References

References

201 http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-4-1.html
202 http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-K.html
203 http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-4-1.html
204 http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-K.html
205 http://www.ada-auth.org/standards/12rm/html/RM-4-1.html
206 http://www.ada-auth.org/standards/12rm/html/RM-K.html

39 Pragmas

39.1 Description

Description

Pragmas[1] control the compiler, i.e. they are compiler directives[2]. They have the standard form of

```
pragma Name (Parameter_List);
```

where the parameter list is optional.

39.2 List of language defined pragmas

List of language defined pragmas

Some pragmas are specially marked:

Ada 2005
This is a new Ada 2005[3] pragma.

Ada 2012
This is a new Ada 2012[4] pragma.

Obsolescent
This is a deprecated pragma and it should not be used in new code.

39.2.1 A – H

- All_Calls_Remote[5]
- Assert[6] (Ada 2005)
- Assertion_Policy[7] (Ada 2005)
- Asynchronous[8] (Obsolescent since Ada 2012)
- Atomic[9] (Obsolescent since Ada 2012)
- Atomic_Components[10] (Obsolescent since Ada 2012)
- Attach_Handler[11] (Obsolescent since Ada 2012)

1 Chapter 39 on page 317
2 http://en.wikipedia.org/wiki/Compiler%20directive
3 Chapter 23 on page 219
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%202012
5 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAll_Calls_Remote
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAssert
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAssertion_Policy
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAsynchronous
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAtomic
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAtomic_Components
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAttach_Handler

- Controlled[12]
- Convention[13] (Obsolescent since Ada 2012)
- CPU[14] (Ada 2012)
- Default_Storage_Pool[15] (Ada 2012)
- Detect_Blocking[16] (Ada 2005)
- Discard_Names[17]
- Dispatching_Domain[18] (Ada 2012)
- Elaborate[19]
- Elaborate_All[20]
- Elaborate_Body[21]
- Export[22] (Obsolescent since Ada 2012)

39.2.2 I – O

- Import[23] (Obsolescent since Ada 2012)
- Independent[24] (Ada 2012)
- Independent_Component[25] (Ada 2012)
- Inline[26] (Obsolescent since Ada 2012)
- Inspection_Point[27]
- Interface[28] (Obsolescent)
- Interrupt_Handler[29] (Obsolescent since Ada 2012)
- Interrupt_Priority[30] (Obsolescent since Ada 2012)
- Linker_Options[31]
- List[32]
- Locking_Policy[33]
- Memory_Size[34] (Obsolescent)
- No_Return[35] (Ada 2005) (Obsolescent since Ada 2012)

12 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FControlled
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FConvention
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCPU
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FDefault_Storage_Pool
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FDetect_Blocking
17 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FDiscard_Names
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FDispatching_Domain
19 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FElaborate
20 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FElaborate_All
21 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FElaborate_Body
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport
24 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FIndependent
25 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FIndependent_Component
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInline
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInspection_Point
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterface
29 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterrupt_Handler
30 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterrupt_Priority
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLinker_Options
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FList
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLocking_Policy
34 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FMemory_Size
35 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNo_Return

- Normalize_Scalars[36]
- Optimize[37]

39.2.3 P – R

- Pack[38] (Obsolescent since Ada 2012)
- Page[39]
- Partition_Elaboration_Policy[40] (Ada 2005)
- Preelaborable_Initialization[41] (Ada 2005)
- Preelaborate[42]
- Priority[43] (Obsolescent since Ada 2012)
- Priority_Specific_Dispatching[44] (Ada 2005)
- Profile[45] (Ada 2005)
- Pure[46]
- Queueing_Policy[47]
- Relative_Deadline[48] (Ada 2005)
- Remote_Call_Interface[49]
- Remote_Types[50]
- Restrictions[51]
- Reviewable[52]

39.2.4 S – Z

- Shared[53] (Obsolescent)
- Shared_Passive[54]
- Storage_Size[55]
- Storage_Unit[56] (Obsolescent)
- Suppress[57]

---

36 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNormalize_Scalars
37 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FOptimize
38 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPack
39 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPage
40 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPartition_Elaboration_Policy
41 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPreelaborable_Initialization
42 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPreelaborate
43 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPriority
44 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPriority_Specific_Dispatching
45 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FProfile
46 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPure
47 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FQueueing_Policy
48 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FRelative_Deadline
49 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FRemote_Call_Interface
50 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FRemote_Types
51 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FRestrictions
52 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FReviewable
53 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShared
54 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShared_Passive
55 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FStorage_Size
56 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FStorage_Unit
57 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSuppress

- System_Name[58] (Obsolescent)
- Task_Dispatching_Policy[59]
- Unchecked_Union[60] (Ada 2005)
- Unsuppress[61] (Ada 2005)
- Volatile[62]
- Volatile_Components[63]

39.3 List of implementation defined pragmas

List of implementation defined pragmas

The following pragmas are not available in all Ada compilers, only in those that had implemented them.
Currently, there are only listed the implementation-defined pragmas of a few compilers. You can help Wikibooks adding[64] specific aspects of other compilers:

GNAT
Implementation defined pragma[65] of the GNAT[66] compiler from AdaCore and FSF.

HP Ada
Implementation defined pragma[67] of the HP Ada[68] compiler (formerly known as "DEC Ada").

ICC
Implementation-defined pragma[69] of the Irvine ICC[70] compiler.

PowerAda
Implementation defined pragma[71] of OC Systems' PowerAda[72].

SPARCompiler
Implementation defined pragma[73] of Sun's SPARCompiler Ada[74].`http://findarticles.com/p/articles/mi_m0EIN/is_1994_Nov_2/ai_15882197`

39.3.1 A – C

- Abort_Defer[75] (GNAT)

58 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSystem_Name`
59 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTask_Dispatching_Policy`
60 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnchecked_Union`
61 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnsuppress`
62 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FVolatile`
63 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FVolatile_Components`
64 `http://en.wikibooks.org/w/index.php?title=Ada_Programming/Pragmas&action=edit`
65 `http://www.adacore.com/wp-content/files/auto_update/gnat-unw-docs/html/gnat_rm_2.html`
66 `http://en.wikibooks.org/wiki/Ada%20Programming%2FGNAT`
67 `http://h71000.www7.hp.com/commercial/ada/ada_lrm.pdf`
68 `http://h71000.www7.hp.com/commercial/ada/ada_index.html`
69 "2.2 ICC-Defined Pragmas", *ICC Ada Implementation Reference — ICC Ada Version 8.2.5 for i960MC Targets*, document version 2.11.4.`http://www.irvine.com/support/general/`
70 `http://www.irvine.com/products.html`
71 `http://www.ocsystems.com/user_guide/powerada/html/powerada-106.html#HEADING106-0`
72 `http://www.ocsystems.com/prod_powerada.html`
73 `http://docs.sun.com/app/docs/doc/802-3641/6i7h8si5i?a=view#F.Implementation-Dependent_Characteristi-2`
74 `http://docs.sun.com/app/docs/coll/15.4`
75 `http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAbort_Defer`

- Ada_83[76] (GNAT)
- Ada_95[77] (GNAT)
- Ada_05[78] (GNAT)
- Ada_2005[79] (GNAT)
- Ada_12[80] (GNAT)
- Ada_2012[81] (GNAT)
- Annotate[82] (GNAT)
- Assume_No_Invalid_Values[83] (GNAT)
- Ast_Entry[84] (GNAT, HP Ada)
- Bit_Pack[85] (SPARCompiler)
- Built_In[86] (SPARCompiler)
- Byte_Pack[87] (SPARCompiler)
- C_Pass_By_Copy[88] (GNAT)
- Call_Mechanism[89] (ICC)
- Canonical_Streams[90] (GNAT)
- Check[91] (GNAT)
- Check_Name[92] (GNAT)
- Check_Policy[93] (GNAT)
- CM_Info[94] (PowerAda)
- Comment[95] (GNAT)
- Common_Object[96] (GNAT, HP Ada)
- Compatible_Calls[97] (ICC)
- Compile_Time_Error[98] (GNAT)
- Compile_Time_Warning[99] (GNAT)
- Complete_Representation[100] (GNAT)

---

76 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAda_83
77 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAda_95
78 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAda_05
79 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAda_2005
80 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAda_12
81 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAda_2012
82 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAnnotate
83 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAssume_No_Invalid_Values
84 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FAst_Entry
85 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FBit_Pack
86 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FBuilt_In
87 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FByte_Pack
88 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FC_Pass_By_Copy
89 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCall_Mechanism
90 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCanonical_Streams
91 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCheck
92 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCheck_Name
93 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCheck_Policy
94 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCM_Info
95 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FComment
96 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCommon_Object
97 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCompatible_Calls
98 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCompile_Time_Error
99 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCompile_Time_Warning
100 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FComplete_Representation

- Complex_Representation[101] (GNAT)
- Component_Alignment[102] (GNAT, HP Ada)
- Compress[103] (ICC)
- Constrain_Private[104] (ICC)
- Convention_Identifier[105] (GNAT)
- CPP_Class[106] (GNAT)
- CPP_Constructor[107] (GNAT)
- CPP_Virtual[108] (GNAT)
- CPP_Vtable[109] (GNAT)

39.3.2 D – H

- Data_Mechanism[110] (ICC)
- Debug[111] (GNAT)
- Debug_Policy[112] (GNAT)
- Delete_Subprogram_Entry[113] (ICC)
- Elaboration_Checks[114] (GNAT)
- Eliminate[115] (GNAT)
- Error[116] (SPARCompiler)
- Export_Exception[117] (GNAT, HP Ada)
- Export_Function[118] (GNAT, HP Ada, SPARCompiler)
- Export_Mechanism[119] (ICC)
- Export_Object[120] (GNAT, HP Ada, SPARCompiler)
- Export_Procedure[121] (GNAT, HP Ada, SPARCompiler)
- Export_Value[122] (GNAT)
- Export_Valued_Procedure[123] (GNAT, HP Ada)
- Extend_System[124] (GNAT)

101 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FComplex_Representation
102 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FComponent_Alignment
103 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCompress
104 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FConstrain_Private
105 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FConvention_Identifier
106 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCPP_Class
107 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCPP_Constructor
108 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCPP_Virtual
109 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FCPP_Vtable
110 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FData_Mechanism
111 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FDebug
112 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FDebug_Policy
113 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FDelete_Subprogram_Entry
114 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FElaboration_Checks
115 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FEliminate
116 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FError
117 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport_Exception
118 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport_Function
119 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport_Mechanism
120 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport_Object
121 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport_Procedure
122 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport_Value
123 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExport_Valued_Procedure
124 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExtend_System

- Extensions_Allowed[125] (GNAT)
- External[126] (GNAT, SPARCompiler)
- External_Name[127] (ICC, SPARCompiler)
- External_Name_Casing[128] (GNAT)
- Fast_Math[129] (GNAT)
- Favor_Top_Level[130] (GNAT)
- Finalize_Storage_Only[131] (GNAT)
- Float_Representation[132] (GNAT, HP Ada)
- Foreign[133] (ICC)
- Generic_Mechanism[134] (ICC)
- Generic_Policy[135] (SPARCompiler)

39.3.3 I – L

- i960_Intrinsic[136] (ICC)
- Ident[137] (GNAT, HP Ada)
- Images[138] (PowerAda)
- Implemented[139], previously named 'Implemented_By_Entry' (GNAT)
- Implicit_Code[140] (SPARCompiler)
- Implicit_Packing[141] (GNAT)
- Import_Exception[142] (GNAT, HP Ada)
- Import_Function[143] (GNAT, HP Ada, SPARCompiler)
- Import_Mechanism[144] (ICC)
- Import_Object[145] (GNAT, HP Ada, SPARCompiler)
- Import_Procedure[146] (GNAT, HP Ada, SPARCompiler)
- Import_Valued_Procedure[147] (GNAT, HP Ada)
- Include[148] (SPARCompiler)

---

125 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExtensions_Allowed
126 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExternal
127 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExternal_Name
128 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FExternal_Name_Casing
129 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FFast_Math
130 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FFavor_Top_Level
131 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FFinalize_Storage_Only
132 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FFloat_Representation
133 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FForeign
134 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FGeneric_Mechanism
135 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FGeneric_Policy
136 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2Fi960_Intrinsic
137 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FIdent
138 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImages
139 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImplemented
140 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImplicit_Code
141 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImplicit_Packing
142 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport_Exception
143 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport_Function
144 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport_Mechanism
145 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport_Object
146 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport_Procedure
147 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FImport_Valued_Procedure
148 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInclude

- Initialize[149] (SPARCompiler)
- Initialize_Scalars[150] (GNAT)
- Inline_Always[151] (GNAT)
- Inline_Generic[152] (GNAT, HP Ada)
- Inline_Only[153] (SPARCompiler)
- Instance_Policy[154] (SPARCompiler)
- Interface_Constant[155] (ICC)
- Interface_Information[156] (PowerAda)
- Interface_Mechanism[157] (ICC)
- Interface_Name[158] (GNAT, HP Ada, ICC, SPARCompiler)
- Interrupt_State[159] (GNAT)
- Invariant[160] (GNAT)
- Keep_Names[161] (GNAT)
- Label[162] (ICC)
- License[163] (GNAT)
- Link_With[164] (GNAT, ICC, SPARCompiler)
- Linker_Alias[165] (GNAT)
- Linker_Constructor[166] (GNAT)
- Linker_Destructor[167] (GNAT)
- Linker_Section[168] (GNAT)
- Long_Float[169] (GNAT: OpenVMS, HP Ada)

39.3.4 M – P

- Machine_Attribute[170] (GNAT)
- Main[171] (GNAT)
- Main_Storage[172] (GNAT, HP Ada)

149 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInitialize
150 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInitialize_Scalars
151 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInline_Always
152 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInline_Generic
153 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInline_Only
154 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInstance_Policy
155 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterface_Constant
156 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterface_Information
157 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterface_Mechanism
158 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterface_Name
159 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInterrupt_State
160 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FInvariant
161 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FKeep_Names
162 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLabel
163 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLicense
164 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLink_With
165 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLinker_Alias
166 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLinker_Constructor
167 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLinker_Destructor
168 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLinker_Section
169 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FLong_Float
170 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FMachine_Attribute
171 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FMain
172 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FMain_Storage

- No_Body[173] (GNAT)
- No_Image[174] (SPARCompiler)
- No_Strict_Aliasing[175] (GNAT)
- No_Suppress[176] (PowerAda)
- No_Reorder[177] (ICC)
- No_Zero[178] (ICC)
- Noinline[179] (ICC)
- Non_Reentrant[180] (SPARCompiler)
- Not_Elaborated[181] (SPARCompiler)
- Not_Null[182] (ICC)
- Obsolescent[183] (GNAT)
- Optimize_Alignment[184] (GNAT)
- Optimize_Code[185] (SPARCompiler)
- Optimize_Options[186] (ICC)
- Ordered[187] (GNAT)
- Parameter_Mechanism[188] (ICC)
- Passive[189] (GNAT, HP Ada, SPARCompiler)
- Persistent_BSS[190] (GNAT)
- Physical_Address[191] (ICC)
- Polling[192] (GNAT)
- Postcondition[193] (GNAT)
- Precondition[194] (GNAT)
- Preserve_Layout[195] (PowerAda)
- Profile_Warnings[196] (GNAT)
- Propagate_Exceptions[197] (GNAT)

---

173 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNo_Body
174 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNo_Image
175 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNo_Strict_Aliasing
176 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNo_Suppress
177 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNo_Reorder
178 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNo_Zero
179 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNoinline
180 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNon_Reentrant
181 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNot_Elaborated
182 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FNot_Null
183 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FObsolescent
184 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FOptimize_Alignment
185 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FOptimize_Code
186 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FOptimize_Options
187 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FOrdered
188 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FParameter_Mechanism
189 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPassive
190 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPersistent_BSS
191 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPhysical_Address
192 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPolling
193 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPostcondition
194 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPrecondition
195 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPreserve_Layout
196 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FProfile_Warnings
197 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPropagate_Exceptions

- Protect_Registers[198] (ICC)
- Protected_Call[199] (ICC)
- Protected_Return[200] (ICC)
- Psect_Object[201] (GNAT, HP Ada)
- Pure_Function[202] (GNAT)
- Put[203] (ICC)
- Put_Line[204] (ICC)

39.3.5 R – S

- Reserve_Registers[205] (ICC)
- Restriction_Warnings[206] (GNAT)
- RTS_Interface[207] (SPARCompiler)
- SCCS_ID[208] (PowerAda)
- Share_Body[209] (SPARCompiler)
- Share_Code[210] (SPARCompiler)
- Share_Generic[211] (GNAT, HP Ada)
- Shareable[212] (ICC)
- Short_Circuit_And_Or[213] (GNAT)
- Short_Descriptors[214] (GNAT)
- Simple_Storage_Pool_Type[215] (GNAT)
- Simple_Task[216] (ICC)
- Source_File_Name[217] (GNAT)
- Source_File_Name_Project[218] (GNAT)
- Source_Reference[219] (GNAT)
- Stack_Size[220] (ICC)
- Static_Elaboration[221] (ICC)

198 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FProtect_Registers
199 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FProtected_Call
200 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FProtected_Return
201 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPsect_Object
202 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPure_Function
203 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPut
204 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FPut_Line
205 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FReserve_Registers
206 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FRestriction_Warnings
207 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FRTS_Interface
208 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSCCS_ID
209 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShare_Body
210 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShare_Code
211 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShare_Generic
212 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShareable
213 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShort_Circuit_And_Or
214 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FShort_Descriptors
215 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSimple_Storage_Pool_Type
216 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSimple_Task
217 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSource_File_Name
218 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSource_File_Name_Project
219 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSource_Reference
220 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FStack_Size
221 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FStatic_Elaboration

- Static_Elaboration_Desired[222] (GNAT)
- Stream_Convert[223] (GNAT)
- Style_Checks[224] (GNAT)
- Subtitle[225] (GNAT)
- Suppress_All[226] (GNAT, HP Ada, PowerAda, SPARCompiler)
- Suppress_Elaboration_Checks[227] (SPARCompiler)
- Suppress_Exception_Locations[228] (GNAT)
- Suppress_Initialization[229] (GNAT)
- System_Table[230] (ICC)

39.3.6 T – Z

- Task_Attributes[231] (SPARCompiler)
- Task_Info[232] (GNAT)
- Task_Name[233] (GNAT)
- Task_Storage[234] (GNAT, HP Ada)
- Test_Case[235] (GNAT)
- Thread_Body[236] (GNAT)
- Thread_Local_Storage[237] (GNAT)
- Time_Slice[238] (GNAT, HP Ada, ICC)
- Time_Slice_Attributes[239] (ICC)
- Title[240] (GNAT, HP Ada)
- Unimplemented_Unit[241] (GNAT)
- Universal_Aliasing[242] (GNAT)
- Universal_Data[243] (GNAT)
- Unmodified[244] (GNAT)

---

222 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FStatic_Elaboration_Desired
223 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FStream_Convert
224 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FStyle_Checks
225 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSubtitle
226 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSuppress_All
227 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSuppress_Elaboration_Checks
228 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSuppress_Exception_Locations
229 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSuppress_Initialization
230 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FSystem_Table
231 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTask_Attributes
232 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTask_Info
233 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTask_Name
234 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTask_Storage
235 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTest_Case
236 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FThread_Body
237 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FThread_Local_Storage
238 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTime_Slice
239 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTime_Slice_Attributes
240 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FTitle
241 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnimplemented_Unit
242 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUniversal_Aliasing
243 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUniversal_Data
244 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnmodified

- Unreferenced[245] (GNAT)
- Unreferenced_Objects[246] (GNAT)
- Unreserve_All_Interrupts[247] (GNAT)
- Unsigned_Literal[248] (ICC)
- Use_VADS_Size[249] (GNAT)
- Validity_Checks[250] (GNAT)
- Warning[251] (SPARCompiler)
- Warnings[252] (GNAT, SPARCompiler)
- Weak_External[253] (GNAT)
- Wide_Character_Encoding[254] (GNAT)

39.4 See also

See also

39.4.1 Wikibook

- Ada Programming[255]
- Ada Programming/Aspects[256]
- Ada Programming/Attributes[257]
- Ada Programming/Keywords[258]

39.4.2 Ada Reference Manual

Ada 83

- Annex 2: Pragmas ^{http://archive.adaic.com/standards/83lrm/html/lrm-2.html}
- Annex B: Predefined Language Pragmas ^{http://archive.adaic.com/standards/83lrm/html/lrm-B.html}

Ada 95

- 2.8 Pragmas[259]
- Annex L: (informative) Language-Defined Pragmas[260]

245 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnreferenced
246 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnreferenced_Objects
247 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnreserve_All_Interrupts
248 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUnsigned_Literal
249 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FUse_VADS_Size
250 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FValidity_Checks
251 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FWarning
252 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FWarnings
253 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FWeak_External
254 http://en.wikibooks.org/wiki/Ada%20Programming%2FPragmas%2FWide_Character_Encoding
255 http://en.wikibooks.org/wiki/Ada%20Programming
256 http://en.wikibooks.org/wiki/Ada%20Programming%2FAspects
257 Chapter 38 on page 305
258 Chapter 35 on page 293
259 http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-2-8.html
260 http://www.adaic.org/resources/add_content/standards/95lrm/ARM_HTML/RM-L.html

Ada 2005

- 2.8 Pragmas[261]
- Annex L: (informative) Language-Defined Pragmas[262]

Ada 2012

- 2.8 Pragmas[263]
- Annex L: (informative) Language-Defined Pragmas[264]

39.5 References

References

261 http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-2-8.html
262 http://www.adaic.org/resources/add_content/standards/05rm/html/RM-2-L.html
263 http://www.ada-auth.org/standards/12rm/html/RM-2-8.html
264 http://www.ada-auth.org/standards/12rm/html/RM-L.html

40 Libraries

40.1 Predefined Language Libraries

Predefined Language Libraries

The library which comes with Ada in general and GNAT[1] in particular. Ada's built in library is quite extensive and well structured. These chapters too are more reference like.

- Standard[2]
- Ada[3]
- Interfaces[4]
- System[5]
- GNAT[6]

40.2 Other Language Libraries

Other Language Libraries

Other libraries which are not part of the standard but freely available.

- Multi Purpose[7]
- Container Libraries[8]
- GUI Libraries[9]
- Distributed Objects[10]
- Database[11]
- Web Programming[12]
- Input/Output[13]

40.3 See also

1 http://en.wikipedia.org/wiki/GNAT
2 Chapter 41 on page 333
3 Chapter 42 on page 337
4 Chapter 43 on page 349
5 Chapter 44 on page 351
6 Chapter 45 on page 353
7 Chapter 46 on page 357
8 Chapter 47 on page 359
9 Chapter 48 on page 361
10 Chapter 49 on page 363
11 Chapter 50 on page 365
12 Chapter 51 on page 371
13 Chapter 52 on page 373

See also

40.3.1 Wikibook

- Ada Programming[14]

40.3.2 Ada Reference Manual

- Annex A (normative) Predefined Language Environment ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A.html`}

40.3.3 Resources

- A collection of Tools and Libraries[15] maintained by the Ada Resource Association.

es:Programación en Ada/Unidades predefinidas[16]

14 `http://en.wikibooks.org/wiki/Ada%20Programming`
15 `http://www.adaic.org/ada-resources/tools-libraries/`
16 `http://es.wikibooks.org/wiki/Programaci%F3n%20en%20Ada%2FUnidades%20predefinidas`

41 Libraries: Standard

The **Standard** package is implicit. This means two things:

1. You do not need to **with** or **use** the package, in fact you cannot (see below). It's always available (except where hidden by a homograph, RM 8.3 (8) ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-8-3.html`}).
2. **Standard** may contain constructs which are not quite legal Ada (like the definitions of `Character` and `Wide_Character`).

A **with** clause mentioning `Standard` references a user-defined package `Standard` that hides the predefined one. So do not do this. However any library unit hidden by a homograph can be made visible again by qualifying its name with `Standard`, like e.g. `Standard.My_Unit`.

41.1 Implementation

Implementation

Since the package `Standard` is very important for portability, here are some examples for various compilers:

- The package Standard[1] from ISO 8652[2].
- The package Standard[3] from GNAT[4].
- The package Standard[5] from Rational Apex[6].
- The package Standard[7] from ObjectAda[8]
- The Standard[9] definitions for AppletMagic[10]

41.2 Portability

Portability

The only mandatory types in Standard are Boolean, Integer and its subtypes, Float, Character, Wide_Character, Wide_Wide_Character, String, Wide_String, Wide_Wide_String, Duration. There is an implementation permission in RM A.1 (51) ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-1.html`} that there may be more integer and floating point types and an implementation advice RM A.1 (52) ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/`}

1 `http://en.wikibooks.org/wiki/%2FRM`
2 `http://en.wikipedia.org/wiki/ISO%208652`
3 `http://en.wikibooks.org/wiki/%2FGNAT`
4 `http://en.wikipedia.org/wiki/GNAT`
5 `http://en.wikibooks.org/wiki/%2FApex`
6 `http://www-306.ibm.com/software/awdtools/developer/ada`
7 `http://en.wikibooks.org/wiki/%2FObjectAda`
8 `http://www.aonix.com/objectada.html`
9 `http://en.wikibooks.org/wiki/%2FAppletMagic`
10 `http://www.sofcheck.com/products/adamagic.html#appletmagic`

RM-A-1.html} about the names to be chosen. There even is no requirement that those additional types must have different sizes. So it is e.g. legal for an implementation to provide two types Long_Integer and Long_Long_Integer which both have the same range and size.
Note that the ranges and sizes of these types can be different in every platform (except of course for Boolean and [[Wide_]Wide_]Character). There is an implementation requirement that the size of type Integer is at least 16 bits, and that of Long_Integer at least 32 bits (if present) RM 3.5.4 (21..22) ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-4.html} . So if you want full portability of your types, do not use types from Standard (except where you must, see below), rather define you own types. A compiler will reject any type declaration whose range it cannot satisfy.
This means e.g. if you need a 64-bit type and find that with your current implementation Standard.Long_Long_Integer is such a type, when porting your program to another implementation, this type may be shorter, but the compiler will not tell you - and your program will most probably crash. However, when you define your own type like

```
type My_Integer_64 is range -(2**63) .. +(2**63 - 1);
```

then, when porting to an implementation that cannot satisfy this range, the compiler will reject your program.
The type Integer is mandatory when you use [[wide] wide] strings or exponentiation x**i. This is why some projects even define their own strings, but this means throwing out the child with the bath tub. Using Integer with strings and exponentiation will normally not lead to portability issues.

41.3 See also

See also

41.3.1 Wikibook

- Ada Programming[11]
- Ada Programming/Libraries#Predefined Language Libraries[12]

41.3.2 Ada Reference Manual

- A.1 The Package Standard ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-1.html}
- 3.5.4 Integer Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-4.html}
- 3.5.7 Floating Point Types ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5-7.html}

11 http://en.wikibooks.org/wiki/Ada%20Programming
12 Chapter 40.1 on page 331

41.3.3 Ada Quality and Style Guide

- 7.1.1 Obsolescent Features ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-1-1.html} – Avoid using the package ASCII
- 7.2.1 Predefined Numeric Types ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/7-2-1.html} – Avoid the predefined numeric types

42 Libraries: Ada

The **Ada** package is only an anchor or namespace for Ada's standard library. Most compilers will not allow you to add new packages to the Ada hierarchy and even if your compiler allows it you should not do so since all package names starting with *Ada.* are reserved for future extensions.

42.1 List of language defined child units

List of language defined child units

The following library units (packages and generic subprograms) are descendents of the package Ada.

Ada 2005

This package is available since Ada 2005[1].

42.1.1 A – C

- Ada.Assertions[2] (Ada 2005)
- Ada.Asynchronous_Task_Control[3]
- Ada.Calendar[4]
- Ada.Calendar.Arithmetic[5] (Ada 2005)
- Ada.Calendar.Formatting[6] (Ada 2005)
- Ada.Calendar.Time_Zones[7] (Ada 2005)
- Ada.Characters[8]
- Ada.Characters.Conversions[9] (Ada 2005)
- Ada.Characters.Handling[10]
- Ada.Characters.Latin_1[11]
- Ada.Command_Line[12]
- Ada.Complex_Text_IO[13] (Ada 2005)

1 Chapter 23 on page 219
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Assertions
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Asynchronous_Task_Control
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Calendar
5 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Calendar.Arithmetic
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Calendar.Formatting
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Calendar.Time_Zones
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Conversions
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Handling
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Latin_1
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Command_Line
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Complex_Text_IO

- Ada.Containers[14] (Ada 2005)
- Ada.Containers.Doubly_Linked_Lists[15] (Ada 2005)
- Ada.Containers.Generic_Array_Sort[16] (Ada 2005 generic procedure)
- Ada.Containers.Generic_Constrained_Array_Sort[17] (Ada 2005 generic procedure)
- Ada.Containers.Hashed_Maps[18] (Ada 2005)
- Ada.Containers.Hashed_Sets[19] (Ada 2005)
- Ada.Containers.Indefinite_Doubly_Linked_Lists[20] (Ada 2005)
- Ada.Containers.Indefinite_Hashed_Maps[21] (Ada 2005)
- Ada.Containers.Indefinite_Hashed_Sets[22] (Ada 2005)
- Ada.Containers.Indefinite_Ordered_Maps[23] (Ada 2005)
- Ada.Containers.Indefinite_Ordered_Sets[24] (Ada 2005)
- Ada.Containers.Indefinite_Vectors[25] (Ada 2005)
- Ada.Containers.Ordered_Maps[26] (Ada 2005)
- Ada.Containers.Ordered_Sets[27] (Ada 2005)
- Ada.Containers.Vectors[28] (Ada 2005)

42.1.2 D – F

- Ada.Decimal[29]
- Ada.Direct_IO[30]
- Ada.Directories[31] (Ada 2005)
- Ada.Directories.Information[32] (Ada 2005)
- Ada.Dispatching[33] (Ada 2005)

14 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Doubly_Linked_Lists
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Generic_Array_Sort
17 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Generic_Constrained_Array_Sort
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Hashed_Maps
19 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Hashed_Sets
20 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Indefinite_Doubly_Linked_Lists
21 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Indefinite_Hashed_Maps
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Indefinite_Hashed_Sets
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Indefinite_Ordered_Maps
24 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Indefinite_Ordered_Sets
25 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Indefinite_Vectors
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Ordered_Maps
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Ordered_Sets
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers.Vectors
29 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Decimal
30 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Direct_IO
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Directories
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Directories.Information
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Dispatching

- Ada.Dispatching.EDF[34] (Ada 2005)
- Ada.Dispatching.Round_Robin[35] (Ada 2005)
- Ada.Dynamic_Priorities[36]
- Ada.Environment_Variables[37] (Ada 2005)
- Ada.Exceptions[38]
- Ada.Execution_Time[39] (Ada 2005)
- Ada.Execution_Time.Timers[40] (Ada 2005)
- Ada.Execution_Time.Group_Budgets[41] (Ada 2005)
- Ada.Finalization[42]
- Ada.Float_Text_IO[43]
- Ada.Float_Wide_Text_IO[44]
- Ada.Float_Wide_Wide_Text_IO[45] (Ada 2005)

42.1.3 G – R

- Ada.Integer_Text_IO[46]
- Ada.Integer_Wide_Text_IO[47]
- Ada.Integer_Wide_Wide_Text_IO[48] (Ada 2005)
- Ada.Interrupts[49]
- Ada.Interrupts.Names[50]
- Ada.IO_Exceptions[51]
- Ada.Numerics[52]
- Ada.Numerics.Complex_Arrays[53] (Ada 2005)
- Ada.Numerics.Complex_Elementary_Functions[54]

---

34 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Dispatching.EDF
35 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Dispatching.Round_Robin
36 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Dynamic_Priorities
37 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Environment_Variables
38 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Exceptions
39 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Execution_Time
40 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Execution_Time.Timers
41 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Execution_Time.Group_Budgets
42 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Finalization
43 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Float_Text_IO
44 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Float_Wide_Text_IO
45 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Float_Wide_Wide_Text_IO
46 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Integer_Text_IO
47 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Integer_Wide_Text_IO
48 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Integer_Wide_Wide_Text_IO
49 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Interrupts
50 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Interrupts.Names
51 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.IO_Exceptions
52 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics
53 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Complex_Arrays
54 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Complex_Elementary_Functions

- Ada.Numerics.Complex_Types[55]
- Ada.Numerics.Discrete_Random[56]
- Ada.Numerics.Elementary_Functions[57]
- Ada.Numerics.Float_Random[58]
- Ada.Numerics.Generic_Complex_Arrays[59] (Ada 2005)
- Ada.Numerics.Generic_Complex_Elementary_Functions[60]
- Ada.Numerics.Generic_Complex_Types[61]
- Ada.Numerics.Generic_Elementary_Functions[62]
- Ada.Numerics.Generic_Real_Arrays[63] (Ada 2005)
- Ada.Numerics.Real_Arrays[64] (Ada 2005)

42.1.4 R – S

- Ada.Real_Time[65]
- Ada.Real_Time.Timing_Events[66] (Ada 2005)
- Ada.Sequential_IO[67]
- Ada.Storage_IO[68]
- Ada.Streams[69]
- Ada.Streams.Stream_IO[70]
- Ada.Strings[71]
- Ada.Strings.Bounded[72]
- Ada.Strings.Bounded.Hash[73] (Ada 2005 generic function)
- Ada.Strings.Fixed[74]
- Ada.Strings.Fixed.Hash[75] (Ada 2005 generic function)

---

55 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Complex_Types
56 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Discrete_Random
57 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Elementary_Functions
58 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Float_Random
59 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Generic_Complex_Arrays
60 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Generic_Complex_Elementary_Functions
61 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Generic_Complex_Types
62 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Generic_Elementary_Functions
63 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Generic_Real_Arrays
64 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Real_Arrays
65 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Real_Time
66 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Real_Time.Timing_Events
67 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Sequential_IO
68 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Storage_IO
69 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Streams
70 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Streams.Stream_IO
71 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings
72 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Bounded
73 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Bounded.Hash
74 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Fixed
75 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Fixed.Hash

- Ada.Strings.Hash[76] (Ada 2005 generic function)
- Ada.Strings.Maps[77]
- Ada.Strings.Maps.Constants[78]
- Ada.Strings.Unbounded[79]
- Ada.Strings.Unbounded.Hash[80] (Ada 2005 generic function)
- Ada.Strings.Wide_Bounded[81]
- Ada.Strings.Wide_Bounded.Wide_Hash[82] (Ada 2005 generic function)
- Ada.Strings.Wide_Fixed[83]
- Ada.Strings.Wide_Fixed.Wide_Hash[84] (Ada 2005 generic function)
- Ada.Strings.Wide_Hash[85] (Ada 2005 generic function)
- Ada.Strings.Wide_Maps[86]
- Ada.Strings.Wide_Maps.Wide_Constants[87]
- Ada.Strings.Wide_Unbounded[88]
- Ada.Strings.Wide_Unbounded.Wide_Hash[89] (Ada 2005 generic function)
- Ada.Strings.Wide_Wide_Bounded[90] (Ada 2005)
- Ada.Strings.Wide_Wide_Bounded.Wide_Wide_Hash[91] (Ada 2005 generic function)
- Ada.Strings.Wide_Wide_Fixed[92] (Ada 2005)
- Ada.Strings.Wide_Wide_Fixed.Wide_Wide_Hash[93] (Ada 2005 generic function)
- Ada.Strings.Wide_Wide_Hash[94] (Ada 2005 generic function)
- Ada.Strings.Wide_Wide_Maps[95] (Ada 2005)
- Ada.Strings.Wide_Wide_Maps.Wide_Wide_Constants[96] (Ada 2005)

---

76 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Hash
77 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Maps
78 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Maps.Constants
79 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Unbounded
80 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Unbounded.Hash
81 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Bounded
82 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Bounded.Wide_Hash
83 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Fixed
84 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Fixed.Wide_Hash
85 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Hash
86 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Maps
87 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Maps.Wide_Constants
88 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Unbounded
89 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Unbounded.Wide_Hash
90 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Bounded
91 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Bounded.Wide_Wide_Hash
92 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Fixed
93 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Fixed.Wide_Wide_Hash
94 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Hash
95 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Maps
96 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Maps.Wide_Wide_Constants

- Ada.Strings.Wide_Wide_Unbounded[97] (Ada 2005)
- Ada.Strings.Wide_Wide_Unbounded.Wide_Wide_Hash[98] (Ada 2005 generic function)
- Ada.Synchronous_Task_Control[99]

42.1.5 T – U

- Ada.Tags[100]
- Ada.Tags.Generic_Dispatching_Constructor[101] (generic function)
- Ada.Task_Attributes[102]
- Ada.Task_Identification[103]
- Ada.Task_Termination[104] (Ada 2005)
- Ada.Text_IO[105]
- Ada.Text_IO.Bounded_IO[106] (Ada 2005)
- Ada.Text_IO.Complex_IO[107]
- Ada.Text_IO.Decimal_IO[108] (Nested package of Ada.Text_IO)
- Ada.Text_IO.Editing[109]
- Ada.Text_IO.Enumeration_IO[110] (Nested package of Ada.Text_IO[111])
- Ada.Text_IO.Fixed_IO[112] (Nested package of Ada.Text_IO)
- Ada.Text_IO.Float_IO[113] (Nested package of Ada.Text_IO)
- Ada.Text_IO.Integer_IO[114] (Nested package of Ada.Text_IO)
- Ada.Text_IO.Modular_IO[115] (Nested package of Ada.Text_IO)
- Ada.Text_IO.Text_Streams[116]
- Ada.Text_IO.Unbounded_IO[117] (Ada 2005)
- Ada.Unchecked_Conversion[118] (generic function)

---

97 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Unbounded
98 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Unbounded.Wide_Wide_Hash
99 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Synchronous_Task_Control
100 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Tags
101 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Tags.Generic_Dispatching_Constructor
102 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Task_Attributes
103 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Task_Identification
104 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Task_Termination
105 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO
106 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Bounded_IO
107 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Complex_IO
108 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Decimal_IO
109 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Editing
110 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Enumeration_IO
111 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO
112 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Fixed_IO
113 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Float_IO
114 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Integer_IO
115 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Modular_IO
116 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Text_Streams
117 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.Unbounded_IO
118 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Unchecked_Conversion

- Ada.Unchecked_Deallocation[119] (generic procedure)

42.1.6 W – Z

- Ada.Wide_Characters[120] (Ada 2005)
- Ada.Wide_Text_IO[121]
- Ada.Wide_Text_IO.Bounded_IO[122] (Ada 2005)
- Ada.Wide_Text_IO.Complex_IO[123]
- Ada.Wide_Text_IO.Decimal_IO[124] (Nested package of Ada.Wide_Text_IO)
- Ada.Wide_Text_IO.Editing[125]
- Ada.Wide_Text_IO.Enumeration_IO[126] (Nested package of Ada.Wide_Text_IO)
- Ada.Wide_Text_IO.Fixed_IO[127] (Nested package of Ada.Wide_Text_IO)
- Ada.Wide_Text_IO.Float_IO[128] (Nested package of Ada.Wide_Text_IO)
- Ada.Wide_Text_IO.Integer_IO[129] (Nested package of Ada.Wide_Text_IO)
- Ada.Wide_Text_IO.Modular_IO[130] (Nested package of Ada.Wide_Text_IO)
- Ada.Wide_Text_IO.Text_Streams[131]
- Ada.Wide_Text_IO.Unbounded_IO[132] (Ada 2005)
- Ada.Wide_Wide_Characters[133] (Ada 2005)
- Ada.Wide_Wide_Text_IO[134] (Ada 2005)
- Ada.Wide_Wide_Text_IO.Bounded_IO[135] (Ada 2005)
- Ada.Wide_Wide_Text_IO.Complex_IO[136] (Ada 2005)
- Ada.Wide_Wide_Text_IO.Decimal_IO[137] (Nested package of Ada.Wide_Wide_-Text_IO)

---

119 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Unchecked_Deallocation
120 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Characters
121 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO
122 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Bounded_IO
123 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Complex_IO
124 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Decimal_IO
125 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Editing
126 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Enumeration_IO
127 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Fixed_IO
128 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Float_IO
129 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Integer_IO
130 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Modular_IO
131 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Text_Streams
132 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.Unbounded_IO
133 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Characters
134 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO
135 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Bounded_IO
136 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Complex_IO
137 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Decimal_IO

- Ada.Wide_Wide_Text_IO.Editing[138] (Ada 2005)
- Ada.Wide_Wide_Text_IO.Enumeration_IO[139] (Nested package of Ada.Wide_Wide_Text_IO)
- Ada.Wide_Wide_Text_IO.Fixed_IO[140] (Nested package of Ada.Wide_Wide_Text_IO)
- Ada.Wide_Wide_Text_IO.Float_IO[141] (Nested package of Ada.Wide_Wide_Text_IO)
- Ada.Wide_Wide_Text_IO.Integer_IO[142] (Nested package of Ada.Wide_Wide_Text_IO)
- Ada.Wide_Wide_Text_IO.Modular_IO[143] (Nested package of Ada.Wide_Wide_Text_IO)
- Ada.Wide_Wide_Text_IO.Text_Streams[144] (Ada 2005)
- Ada.Wide_Wide_Text_IO.Unbounded_IO[145] (Ada 2005)

42.2 List of implementation defined child units

List of implementation defined child units

The Reference Manual allows compiler vendors to add extensions to the Standard Libraries. However, these extensions cannot be directly childs of the package Ada, only grandchilds -- for example Ada.Characters.Latin_9 .
Currently, only the implementation defined library units of the GNAT[146] compiler are listed here. You can help Wikibooks by adding[147] implementation dependent packages of other compilers:

GNAT
Extended package implemented by GNAT[148].

ObjectAda
Extended package implemented by ObjectAda.

APEX
Extended package implemented by IBM/Rational APEX.

138 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Editing
139 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Enumeration_IO
140 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Fixed_IO
141 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Float_IO
142 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Integer_IO
143 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Modular_IO
144 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Text_Streams
145 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.Unbounded_IO
146 http://en.wikipedia.org/wiki/GNAT
147 http://en.wikibooks.org/w/index.php?title=Ada_Programming/Libraries/Ada&action=edit
148 http://gcc.gnu.org/onlinedocs/gnat_rm/The-GNAT-Library.html

42.2.1 A – K

- Ada.Characters.Latin_9[149] (GNAT)
- Ada.Characters.Wide_Latin_1[150] (GNAT)
- Ada.Characters.Wide_Latin_9[151] (GNAT)
- Ada.Characters.Wide_Wide_Latin_1[152] (GNAT)
- Ada.Characters.Wide_Wide_Latin_9[153] (GNAT)
- Ada.Command_Line.Environment[154] (GNAT)
- Ada.Command_Line.Remove[155] (GNAT)
- Ada.Direct_IO.C_Streams[156] (GNAT)
- Ada.Exceptions.Is_Null_Occurrence[157] (GNAT child function)
- Ada.Exceptions.Traceback[158] (GNAT)

42.2.2 L – Q

- Ada.Long_Float_Text_IO[159] (GNAT)
- Ada.Long_Float_Wide_Text_IO[160] (GNAT)
- Ada.Long_Integer_Text_IO[161] (GNAT)
- Ada.Long_Integer_Wide_Text_IO[162] (GNAT)
- Ada.Long_Long_Float_Text_IO[163] (GNAT)
- Ada.Long_Long_Float_Wide_Text_IO[164] (GNAT)
- Ada.Long_Long_Integer_Text_IO[165] (GNAT)
- Ada.Long_Long_Integer_Wide_Text_IO[166] (GNAT)
- Ada.Numerics.Long_Complex_Elementary_Functions[167] (GNAT)

---

149 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Latin_9
150 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Wide_Latin_1
151 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Wide_Latin_9
152 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Wide_Wide_Latin_1
153 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Characters.Wide_Wide_Latin_9
154 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Command_Line.Environment
155 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Command_Line.Remove
156 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Direct_IO.C_Streams
157 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Exceptions.Is_Null_Occurrence
158 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Exceptions.Traceback
159 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Float_Text_IO
160 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Float_Wide_Text_IO
161 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Integer_Text_IO
162 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Integer_Wide_Text_IO
163 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Long_Float_Text_IO
164 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Long_Float_Wide_Text_IO
165 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Long_Integer_Text_IO
166 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Long_Long_Integer_Wide_Text_IO
167 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Long_Complex_Elementary_Functions

- Ada.Numerics.Long_Complex_Types[168] (GNAT)
- Ada.Numerics.Long_Elementary_Functions[169] (GNAT)
- Ada.Numerics.Long_Long_Complex_Elementary_Functions[170] (GNAT)
- Ada.Numerics.Long_Long_Complex_Types[171] (GNAT)
- Ada.Numerics.Long_Long_Elementary_Functions[172] (GNAT)
- Ada.Numerics.Short_Complex_Elementary_Functions[173] (GNAT)
- Ada.Numerics.Short_Complex_Types[174] (GNAT)
- Ada.Numerics.Short_Elementary_Functions[175] (GNAT)

42.2.3 R – Z

- Ada.Sequential_IO.C_Streams[176] (GNAT)
- Ada.Short_Float_Text_IO[177] (GNAT)
- Ada.Short_Float_Wide_Text_IO[178] (GNAT)
- Ada.Short_Integer_Text_IO[179] (GNAT)
- Ada.Short_Integer_Wide_Text_IO[180] (GNAT)
- Ada.Short_Short_Integer_Text_IO[181] (GNAT)
- Ada.Short_Short_Integer_Wide_Text_IO[182] (GNAT)
- Ada.Streams.Stream_IO.C_Streams[183] (GNAT)
- Ada.Strings.Unbounded.Text_IO[184] (GNAT)
- Ada.Strings.Wide_Unbounded.Wide_Text_IO[185] (GNAT)

---

168 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Long_Complex_Types

169 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Long_Elementary_Functions

170 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Long_Long_Complex_Elementary_Functions

171 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Long_Long_Complex_Types

172 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Long_Long_Elementary_Functions

173 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Short_Complex_Elementary_Functions

174 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Short_Complex_Types

175 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Numerics.Short_Elementary_Functions

176 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Sequential_IO.C_Streams

177 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Short_Float_Text_IO

178 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Short_Float_Wide_Text_IO

179 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Short_Integer_Text_IO

180 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Short_Integer_Wide_Text_IO

181 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Short_Short_Integer_Text_IO

182 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Short_Short_Integer_Wide_Text_IO

183 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Streams.Stream_IO.C_Streams

184 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Unbounded.Text_IO

185 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Unbounded.Wide_Text_IO

- Ada.Strings.Wide_Wide_Unbounded.Wide_Wide_Text_IO[186] (GNAT)
- Ada.Text_IO.C_Streams[187] (GNAT)
- Ada.Wide_Text_IO.C_Streams[188] (GNAT)
- Ada.Wide_Wide_Text_IO.C_Streams[189] (GNAT)

42.3 See also

See also

42.3.1 Wikibook

- Ada Programming[190]
- Ada Programming/Libraries[191]
- Ada Programming/Libraries/Standard[192]
- Ada Programming/Libraries/System[193]
- Ada Programming/Libraries/Interfaces[194]

42.3.2 Ada Reference Manual

A.2 The Package Ada ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-2.html}

186 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Strings.Wide_Wide_Unbounded.Wide_Wide_Text_IO
187 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Text_IO.C_Streams
188 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Text_IO.C_Streams
189 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Wide_Wide_Text_IO.C_Streams
190 http://en.wikibooks.org/wiki/Ada%20Programming
191 Chapter 40 on page 331
192 Chapter 41 on page 333
193 Chapter 44 on page 351
194 Chapter 43 on page 349

43 Libraries: Interfaces

The Interfaces package helps in interfacing with other programming languages. Ada is one of the few languages where interfacing with other languages is part of the language standard. The language standard defines the interfaces for C[1], Cobol[2] and Fortran[3]. Of course any implementation might define further interfaces — GNAT[4] for example defines an interface to C++[5].
Interfacing with other languages is actually provided by **pragma** , **pragma** and **pragma**

43.1 Child Packages

Child Packages

- Interfaces.C[6]
  - Interfaces.C.Extensions[7] (GNAT)
  - Interfaces.C.Pointers[8]
  - Interfaces.C.Streams[9] (GNAT)
  - Interfaces.C.Strings[10]
- Interfaces.CPP[11] (GNAT)
- Interfaces.COBOL[12]
- Interfaces.Fortran[13]
- Interfaces.OS2Lib[14] (GNAT, OS/2)
  - Interfaces.OS2Lib.Errors[15] (GNAT, OS/2)
  - Interfaces.OS2Lib.Synchronization[16] (GNAT, OS/2)
  - Interfaces.OS2Lib.Threads[17] (GNAT, OS/2)
- Interfaces.Packed_Decimal[18] (GNAT)

---

1 http://en.wikibooks.org/wiki/C%20Programming
2 http://en.wikibooks.org/wiki/COBOL
3 http://en.wikibooks.org/wiki/Programming%3AFortran
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FGNAT
5 http://en.wikibooks.org/wiki/C%2B%2B%20Programming
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.C
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.C.Extensions
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.C.Pointers
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.C.Streams
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.C.Strings
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.CPP
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.COBOL
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.Fortran
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.OS2Lib
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.OS2Lib.Errors
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.OS2Lib.Synchronization
17 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.OS2Lib.Threads
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.Packed_Decimal

- Interfaces.VxWorks[19] (GNAT, VxWorks)
  - Interfaces.VxWorks.IO[20] (GNAT, VxWorks)

43.2 See also

See also

43.2.1 Wikibook

- Ada Programming[21]
- Ada Programming/Libraries[22]
- Ada Programming/Libraries/Standard[23]
- Ada Programming/Libraries/Ada[24]
- Ada Programming/Libraries/System[25]

43.2.2 Ada Reference Manual

Ada 95

- Annex B Interface to Other Languages ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B.html}
- B.2 The Package Interfaces ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-2.html}

Ada 2005

- Annex B Interface to Other Languages ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B.html}
- B.2 The Package Interfaces ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-B-2.html}

19 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.VxWorks
20 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FInterfaces.VxWorks.IO
21 http://en.wikibooks.org/wiki/Ada%20Programming
22 Chapter 40 on page 331
23 Chapter 41 on page 333
24 Chapter 42 on page 337
25 Chapter 44 on page 351

44 Libraries: System

45 Libraries: GNAT

The GNAT package hierarchy defines several units for general purpose programming provided by the GNAT compiler. It is distributed along with the compiler and uses the same license.
GNAT-4-ObjectAda[1] is a project for porting the GNAT library to the ObjectAda compiler.

45.1 Child packages

Child packages

- GNAT.Array_Split[2]
- GNAT.AWK[3]
- GNAT.Bounded_Buffers[4]
- GNAT.Bounded_Mailboxes[5]
- GNAT.Bubble_Sort[6]
- GNAT.Bubble_Sort_A[7]
- GNAT.Bubble_Sort_G[8]
- GNAT.Calendar[9]
  - GNAT.Calendar.Time_IO[10]
- GNAT.Case_Util[11]
- GNAT.CGI[12]
  - GNAT.CGI.Cookie[13]
  - GNAT.CGI.Debug[14]
- GNAT.Command_Line[15]
- GNAT.Compiler_Version[16]
- GNAT.CRC32[17]
- GNAT.Ctrl_C[18]

1 http://sourceforge.net/projects/gnat4oa
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Array_Split
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.AWK
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Bounded_Buffers
5 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Bounded_Mailboxes
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Bubble_Sort
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Bubble_Sort_A
8 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Bubble_Sort_G
9 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Calendar
10 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Calendar.Time_IO
11 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Case_Util
12 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.CGI
13 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.CGI.Cookie
14 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.CGI.Debug
15 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Command_Line
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Compiler_Version
17 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.CRC32
18 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Ctrl_C

- GNAT.Current_Exception[19]
- GNAT.Debug_Pools[20]
- GNAT.Debug_Utilities[21]
- GNAT.Directory_Operations[22]
  - GNAT.Directory_Operations.Iteration[23]
- GNAT.Dynamic_HTables[24]
- GNAT.Dynamic_Tables[25]
- GNAT.Exception_Actions[26]
- GNAT.Exceptions[27]
- GNAT.Exception_Traces[28]
- GNAT.Expect[29]
- GNAT.Float_Control[30]
- GNAT.Heap_Sort[31]
- GNAT.Heap_Sort_A[32]
- GNAT.Heap_Sort_G[33]
- GNAT.HTable[34]
- GNAT.IO[35]
- GNAT.IO_Aux[36]
- GNAT.Lock_Files[37]
- GNAT.MD5[38]
- GNAT.Memory_Dump[39]
- GNAT.Most_Recent_Exception[40]
- GNAT.OS_Lib[41]
- GNAT.Perfect_Hash_Generators[42]

---

19 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Current_Exception
20 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Debug_Pools
21 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Debug_Utilities
22 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Directory_Operations
23 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Directory_Operations.Iteration
24 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Dynamic_HTables
25 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Dynamic_Tables
26 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Exception_Actions
27 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Exceptions
28 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Exception_Traces
29 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Expect
30 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Float_Control
31 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Heap_Sort
32 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Heap_Sort_A
33 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Heap_Sort_G
34 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.HTable
35 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.IO
36 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.IO_Aux
37 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Lock_Files
38 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.MD5
39 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Memory_Dump
40 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Most_Recent_Exception
41 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.OS_Lib
42 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Perfect_Hash_Generators

- GNAT.Regexp[43]
- GNAT.Registry[44]
- GNAT.Regpat[45]
- GNAT.Secondary_Stack_Info[46]
- GNAT.Semaphores[47]
- GNAT.Signals[48]
- GNAT.Sockets[49] (GNAT.Sockets examples[50])
  - GNAT.Sockets.Constants[51]
  - GNAT.Sockets.Linker_Options[52]
  - GNAT.Sockets.Thin[53]
- GNAT.Source_Info[54]
- GNAT.Spelling_Checker[55]
- GNAT.Spitbol[56]
  - GNAT.Spitbol.Patterns[57]
  - GNAT.Spitbol.Table_Boolean new[58]
  - GNAT.Spitbol.Table_Integer[59]
  - GNAT.Spitbol.Table_VString new[60]
- GNAT.Strings[61]
- GNAT.String_Split[62]
- GNAT.Table[63]
- GNAT.Task_Lock[64]
- GNAT.Threads[65]
- GNAT.Traceback[66]

---

43 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Regexp
44 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Registry
45 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Regpat
46 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Secondary_Stack_Info
47 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Semaphores
48 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Signals
49 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Sockets
50 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Sockets_examples
51 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Sockets.Constants
52 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Sockets.Linker_Options
53 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Sockets.Thin
54 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Source_Info
55 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Spelling_Checker
56 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Spitbol
57 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Spitbol.Patterns
58 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Spitbol.Table_Boolean%20new
59 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Spitbol.Table_Integer
60 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Spitbol.Table_VString%20new
61 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Strings
62 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.String_Split
63 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Table
64 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Task_Lock
65 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Threads
66 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Traceback

  - GNAT.Traceback.Symbolic[67]
- GNAT.Wide_String_Split[68]

45.2 See also

See also

45.2.1 External links

- The GNAT Library[69]

45.2.2 Wikibook

- Ada Programming[70]
- Ada Programming/Libraries[71]

67 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Traceback.Symbolic
68 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGNAT.Wide_String_Split
69 http://gcc.gnu.org/onlinedocs/gnat_rm/The-GNAT-Library.html
70 http://en.wikibooks.org/wiki/Ada%20Programming
71 Chapter 40 on page 331

46 Libraries: Multi-Purpose

AdaCL, Ada Class Library[1]
Filtering of text files, string tools, process control, command line parsing, CGI, garbage collector, components.

Matreshka[2]
Core components for information system development: Unicode support (case conversions and folding, collation, normalization); regular expression engine; XML processor; FastCGI, SQL database access.

paraffin[3]
"A suite of Ada 2005 generics to facilitate iterative and recursive parallelism".[4] Features include load-balancing and monitoring of stacks.

46.1 See also

See also

46.1.1 Wikibook

- Ada Programming[6]
- Ada Programming/Libraries[7]

46.1.2 Ada Reference Manual

-- does not apply --

46.1.3 References

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FAdaCL
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FMatreshka
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FParaffin
4 Iterative and recursive parallelism generics for Ada 2005 [5]. . Retrieved 2012-08-28
6 http://en.wikibooks.org/wiki/Ada%20Programming
7 Chapter 40 on page 331

47 Libraries: Container

The following Libraries help you store and manage objects inside container classes:

Booch Components[1]
the most complete of all container class libraries (at least when used with AdaCL, Ada Class Library[2]).

AdaCL, Ada Class Library[3]
A Booch Components[4] extension pack for storing indefinite objects.

Charles[5]
Build on the C++ STL and therefore very easy to learn for C++ developers.

AI302[6]
Proof of concept for Ada.Containers[7]

Ada.Containers[8]
This language feature is only available in Ada 2005

Stephe's Ada Library[9]
dynamic arrays, lists, trees

47.1 See also

See also

47.1.1 Wikibook

- Ada Programming[10]
- Ada Programming/Libraries[11]

47.1.2 Ada Reference Manual

- A.18.1 The Package Containers ^{`http://www.adaic.org/resources/add_content/standards/05rm/html/RM-A-18-1.html`}

1 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FContainer%2FBooch`
2 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FAdaCL`
3 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FAdaCL`
4 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FContainer%2FBooch`
5 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FContainer%2FCharles`
6 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FContainer%2FAI302`
7 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers`
8 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FAda.Containers`
9 `http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FSAL`
10 `http://en.wikibooks.org/wiki/Ada%20Programming`
11 Chapter 40 on page 331

48 Libraries: GUI

The following libraries can be used to make Graphical User Interfaces:

CLAW[1]
Commercial GUI toolkit for Windows. Introductory Edition[2] is distributed under GMGPL[3].

GtkAda[4]
Binding to the popular GTK+ toolkit.

GWindows[5]
RAD GUI Development Framework for Windows.

Qt4Ada[6]
An Ada2005 binding to Qt4. Under CeCILL license V2.

QtAda[7]
An Ada2005 binding to the Qt libraries and associated tools. Under GPL and GMGPL[8] (commercially supported) licenses.

libAgar[9]
Ada bindings for the libagar[10] OpenGL GUI library (BSD license).

TASH[11]
TclAdaSHell, An Ada binding to Tcl/Tk. GPL with "Linking Exception".

48.1 See also

See also

48.1.1 Wikibook

- Ada Programming[12]
- Ada Programming/Libraries[13]

48.1.2 Ada Reference Manual

-- does not apply --

1 http://www.rrsoftware.com/html/prodinf/claw/claw.htm
2 http://www.adapower.com/adapower1/claw/
3 http://en.wikipedia.org/wiki/GMGPL
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGUI%2FGtkAda
5 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGUI%2FGWindows
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGUI%2FQt4Ada
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGUI%2FQtAda
8 http://en.wikipedia.org/wiki/GMGPL
9 http://wiki.libagar.org/wiki/Ada_bindings
10 http://libagar.org/
11 http://sourceforge.net/projects/tcladashell/
12 http://en.wikibooks.org/wiki/Ada%20Programming
13 Chapter 40 on page 331

48.1.3 External Links

- adapower.com - Links to tools and Bindings for GUI Applications[14]
- adapower.com - Examples of programming GUIs in Ada[15]

14 http://www.adapower.com/index.php?Command=Class&ClassID=AdaGUI&Title=Ada+GUI

15 http://www.adapower.com/index.php?Command=Class&ClassID=GUIExamples&Title=GUI+Examples

49 Libraries: Distributed Systems

The following Libraries help you in Distributed programming:
GLADE[1]
A full implementation of the Ada Annex E: Distributed Systems[2]
PolyORB[3]
A CORBA[4] and Annex E: Distributed Systems[5] implementation.

49.1 See also

See also

49.1.1 Wikibook

- Ada Programming[6]
- Ada Programming/Libraries[7]
- Programming:CORBA[8]

49.1.2 Ada Reference Manual

- Annex E (normative) Distributed Systems ^{http://www.adaic.org/resources/add_content/standards/05rm/html/RM-E.html}

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FDistributed%2FGLADE
2 http://www.adaic.org/standards/95lrm/html/RM-E.html
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FDistributed%2FPolyORB
4 http://en.wikibooks.org/wiki/Programming%3ACORBA
5 http://www.adaic.org/standards/95lrm/html/RM-E.html
6 http://en.wikibooks.org/wiki/Ada%20Programming
7 Chapter 40 on page 331
8 http://en.wikibooks.org/wiki/Programming%3ACORBA

50 Libraries: Databases

The following libraries help you in Database programming:

Library	License	Interbase Firebird	MySQL	ODBC	Oracle	PostgreSQL	SQLite 3	Sybase	Other database	Binary packages	Notes
APQ[1]	GMGPL	—	**Yes**	**Yes**	—	**Yes**	—	**Yes**	—	Debian[2]	Thread-safe connection pools
GNADE (GNu Ada Database Environment)[3]	GMGPL	—	**3.x, 4.x**	**Yes**	—	**Yes**	**Yes**	—	—	Debian[4]	Embedded SQL preprocessor
gnadelite[5]	?	?	?	?	?	?	**Yes**	?	?	?	

1 http://framework.kow.com.br
2 http://packages.qa.debian.org/a/apq.html
3 http://gnade.sourceforge.net
4 http://packages.qa.debian.org/g/gnade.html
5 http://repo.or.cz/w/gnadelite.git

GNAT-COLL (database interface module)[6]	GPL/GMGPL	—	—	—	—	**Yes**	**Yes**	—	—	?	gnatcoll_db2ada generates thick Ada bindings to a specified database schema. Requires Ada 2005.
GWindows[7]	?	?	?	**Yes**	?	?	?	?	?	?	Windows only?
Matreshka SQL[8]	BSD	**Yes**	**Yes**	—	**Yes**	**Yes**	**Yes**	—	—	OpenSUSE[9], Fedora[10]	

6 http://www.adacore.com/wp-content/files/auto_update/gnatcoll-docs/gnatcoll.html#Database-interface
7 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FDatabase%2FGWindows
8 http://adaforge.qtada.com/cgi-bin/tracker.fcgi/matreshka/wiki
9 http://www.opensuse.org
10 http://www.fedoraproject.org

ODBC[11]	GPL	?	?	**Yes**	?	?	?	?	?	none	Windows only. Depends on Win32Ada. Very simple.
QtAda (QtSql module)[12]	GPL/GMGPL	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	IBM DB/2, SQLite2	Microsoft Windows	Binding to Qt: requires C++

11 http://pobry.blogspot.fr/p/ada-contributions.html
12 http://www.qtada.com/

Simple Components for Ada[13]	GMGPL	—	—	**Yes**	—	—	**Yes**	—	—	Fedora, Debian	SQLite bindings are intended for static linking with the SQLite3 amalgation. ODBC bindings support 32- and 64-bit platforms
SOCI-Ada[14]	Boost	—	**Yes**	—	**Yes**	**Yes**	—	—	—	?	Requires C++.
SQLite3-Ada[15]	Public domain	—	—	—	—	—	**Yes**	—	—	?	

13 http://www.dmitry-kazakov.de/ada/components.htm#SQLite
14 http://www.inspirel.com/soci-ada/
15 http://coreland.ath.cx/code/sqlite3-ada

51 Libraries: Web

The following libraries help you in Internet or Web programming:
AdaCL, Ada Class Library[1]
Powerful CGI implementation.
XML/Ada[2]
XML and Unicode support.
AWS[3]
A full-featured Web-Server.
Matreshka[4]
FastCGI, XML, Unicode and localization support.

51.1 See also

See also

51.1.1 Wikibook

- Ada Programming[5]
- Ada Programming/Libraries[6]

51.1.2 Ada Reference Manual

-- does not apply --

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FAdaCL
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FIO%2FXML%2FAda
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FWeb%2FAWS
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FMatreshka
5 http://en.wikibooks.org/wiki/Ada%20Programming
6 Chapter 40 on page 331

52 Libraries: Input Output

The following libraries help you when doing input/output[1]:

AdaCL, Ada Class Library[2]
A multipurpose library featuring filtering of text files, string I/O, command line parsing, etc.

XML/Ada[3]
XML and Unicode support.

Matreshka[4]
SAX-style XML reader and writer. Supports XML1.0 (Fifth Edition), XML1.1 (Second Edition), Namespaces in XML and XML Base Specifications. Strings, files and sockets can be used as input source in both blocking and non-blocking modes. Full Unicode support and many text codecs is provided also.

52.1 See also

See also

52.1.1 Wikibook

- Ada Programming[5]
- Ada Programming/Libraries[6]

52.1.2 Ada Reference Manual

-- does not apply --

1 Chapter 18 on page 147
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FAdaCL
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FIO%2FXML%2FAda
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FMultiPurpose%2FMatreshka
5 http://en.wikibooks.org/wiki/Ada%20Programming
6 Chapter 40 on page 331

53 Platform Support

Ada is known to be very portable, but there is sometimes a necessity of using a specific platform feature. For that matter, there are some non-standard libraries.

- Linux[1]
- Windows[2]
- POSIX systems[3]
- Virtual machines[4]
  - Java[5]
  - .NET[6]

53.1 See also

See also

53.1.1 Wikibook

- Ada Programming[7]

53.1.2 Ada Reference Manual

-- does not apply --

53.1.3 Ada Quality and Style Guide

- Chapter 7: Portability ^{http://www.adaic.org/resources/add_content/docs/95style/html/sec_7/}

1 Chapter 54 on page 377
2 Chapter 55 on page 379
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FPOSIX
4 Chapter 56 on page 381
5 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FVM%2FJava
6 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FVM%2FdotNET
7 http://en.wikibooks.org/wiki/Ada%20Programming

54 Platform: Linux

The following libraries help you when you target the Linux Platform.

Florist[1]
POSIX.5 binding. It will let you perform Linux system calls in the POSIX subset.

Ncurses[2]
text terminal library.

Texttools[3]
ncurses-based library for the Linux console.

GtkAda[4]
GUI library (actually multiplatform).

54.1 See also

See also

54.1.1 Wikibook

- Ada Programming[5]
- Ada Programming/Libraries[6]

54.1.2 Ada Reference Manual

-- does not apply --

54.1.3 External resources

- The Big Online Book of Linux Ada Programming[7]
- Ada in Debian GNU/Linux[8], slides suitable for a 50minute presentation, by Ludovic Brenta[9].

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FPOSIX
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FNcurses
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FTexttools
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FLibraries%2FGUI%2FGtkAda
5 http://en.wikibooks.org/wiki/Ada%20Programming
6 Chapter 40 on page 331
7 http://www.pegasoft.ca/resources/boblap/book.html
8 http://www.cs.kuleuven.be/~dirk/ada-belgium/events/06/060226-fosdem-4-ada-in-debian.pdf
9 http://en.wikibooks.org/wiki/User%3ALudovic%20Brenta

55 Platform: Windows

The following Libraries and Tools help you when you target the MS-Windows Platform.

GWindows[1]
Win32 binding

CLAW[2]
Another Win32 binding that works with any Ada 95 compiler. An introductory edition is available free of charge for non-commercial use.

GNATCOM[3]
COM/DCOM/ActiveX binding

GNAVI[4]
Visual RAD[5] (Rapid application development[6]) Development environment

/Console/[7]
Libraries for console I/O.

/Visual C++ - GNAT interface/[8]
Guide for calling Ada functions from C++ using GNAT and Visual C++.

55.1 See also

See also

55.1.1 Wikibook

- Ada Programming[9]
- Ada Programming/Libraries[10]

55.1.2 Ada Reference Manual

-- does not apply --

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FWindows%2Fwin32binding
2 http://www.rrsoftware.com/html/prodinf/claw/claw.htm
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FWindows%2Fwin32binding
4 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FWindows%2Fwin32binding
5 http://en.wikipedia.org/wiki/Rapid_application_development
6 http://en.wikipedia.org/wiki/Rapid%20application%20development
7 http://en.wikibooks.org/wiki/%2FConsole%2F
8 http://en.wikibooks.org/wiki/%2FVisual%20C%2B%2B%20-%20GNAT%20interface%2F
9 http://en.wikibooks.org/wiki/Ada%20Programming
10 Chapter 40 on page 331

56 Platform: Virtual Machines

The following tools help you when you target a virtual machine.

Java[1]
Programming Ada 95 for Java's JVM (JGnat, AppletMagic)

.NET[2]
Programming Ada for the .NET Platform (GNAT Pro .NET, A#)

56.1 See also

See also

56.1.1 Wikibook

- Ada Programming[3]
- Ada Programming/Libraries[4]

56.1.2 Ada Reference Manual

-- does not apply --

1 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FVM%2FJava
2 http://en.wikibooks.org/wiki/Ada%20Programming%2FPlatform%2FVM%2FdotNET
3 http://en.wikibooks.org/wiki/Ada%20Programming
4 Chapter 40 on page 331

57 Portals

57.1 Forges of open-source projects

Forges of open-source projects

SourceForge[1]

Currently there are 200 Ada projects hosted at SourceForge — including the example programs for Ada Programming[2] wikibook.

GitHub[3]

A source code repository based on Git with many recent developments.

Ada-centric forges

There are some Ada-centric forges hosted by Ada associations and individuals:

- `http://forge.ada-ru.org`
- `http://www.ada-france.org:8081`
- `http://codelabs.ch`
- `http://scm.ada.cx`

BerliOS[4]

57.2 Directories of freely available tools and libraries

Directories of freely available tools and libraries

Ada Information Clearinghouse — Free Tools and Libraries[5]

Oloh (language summary[6]**, ada tag**[7]**, language search**[8]**)**

Oloh is a directory of Open Source projects. Its main features are source code analysis of public repositories and public reviews of projects.

Freecode[9]

Freecode, formerly Freshmeat.net, is a software directory where developers can register their projects and users find interesting software. Although the content is somewhat redundant to other portals, some projects are exclusively listed here.

57.3 Collections of Ada source code

1 `http://sourceforge.net/directory/language:ada/`
2 `https://sourceforge.net/projects/wikibook-ada`
3 `https://github.com/languages/Ada`
4 `http://developer.berlios.de/softwaremap/trove_list.php?form_cat=52`
5 `http://www.adaic.org/ada-resources/tools-libraries/`
6 `http://www.ohloh.net/languages/21`
7 `http://www.ohloh.net/tags/ada`
8 `http://www.ohloh.net/p?page=3&q=language%3Aada&sort=relevance`
9 `http://freecode.com/tags/ada?sort=vitality&with=&without=`

Collections of Ada source code

AdaBasis[10]

AdaBasis consists of about 560 MB of public domain source code and documents, mainly taken from the Public Ada Library (PAL). The software has been classified and is presented in a hierarchical manner, separated in different application domains, and, for some domains, with an additional multi-faceted searching facility.

The intent is to provide students, teachers and researchers with a large collection of reusable Ada components and systems for use in language and software engineering courses.

AdaBasis was set up by the Programming Languages Group of the Institut für Informatik at the University of Stuttgart, Germany. They plan to enlarge the library in the future, and welcome free public domain contributions. For more informations or to make suggestions please contact adabasis@informatik.uni-stuttgart.de[11]

The Public Ada Library (PAL)[12]

The PAL is a library of Ada and VHDL software, information, and courseware that contains over 1 BILLION bytes of material (mainly in compressed form). All items in the PAL have been released to the public with unlimited distribution, and, in most cases (the exceptions are shareware), the items are freeware.

[**ftp**

//ftp.cs.kuleuven.ac.be/pub/Ada-Belgium/cdrom/index.html Ada and Software Engineering Library Version 2 (ASE2)] : *The ASE2 Library contains over 1.1GB of material on Ada and Software Engineering assembled through a collaboration with over 60 organizations.* Walnut Creek CDROM once sold copies of this library. Nowadays it is no longer maintained but is still hosted in the Ada Belgium FTP server. It may contain useful resources, but it is highly redundant with other libraries.

AdaPower[13]

A directory and collection of Ada tools and resources.

57.4 See also

See also

57.4.1 Wikibook

- Ada Programming[14]
- Ada Programming/Tutorials[15]
- Ada Programming/Wikis[16]

57.4.2 Ada Reference Manual

-- does not apply --

10 http://www.iste.uni-stuttgart.de/ps/adabasis.html
11 mailto:adabasis@informatik.uni-stuttgart.de
12 http://www2.informatik.uni-stuttgart.de/iste/ps/ada-software/html/PAL.html
13 http://www.adapower.com/index.php?Command=Packages&Title=Packages+for+Reuse
14 http://en.wikibooks.org/wiki/Ada%20Programming
15 Chapter 58 on page 387
16 http://en.wikibooks.org/wiki/Ada%20Programming%2FWikis

57.4.3 Ada Quality and Style Guide

-- does not apply --

58 Tutorials

This page contains a list of other Ada tutorials on the Net.

1. Ada Programming[1], available on Wikibooks, is currently the only tutorial based on the Ada 2005[2] standard and currently being updated to Ada 2012[3].
2. Lovelace[4] is a free (no-charge), self-directed Ada 95 tutorial available on the World Wide Web (WWW). Lovelace assumes that the user already knows another algorithmic programming language, such as C, C++, or Pascal. Lovelace is interactive and contains many short sections, most of which end with a question to help ensure that users understand the material. Lovelace can be used directly from the WWW, downloaded, or run from CD-ROM. Lovelace was developed by David A. Wheeler.
3. AdaTutor[5] is an interactive Ada 95 tutorial distributed as a public-domain Ada program. A web edition[6] of the tutorial is also available.
4. The Ada-95: A guide for C and C++ programmers[7] is a short hypertext tutorial for programmers who have a C or C++ style programming language background. It was written by Simon Johnston, with some additional text by Tucker Taft. PDF edition[8].
5. Dale Stanbrough's Introduction[9] is a set of notes that provide a simple introduction to Ada. This material has been used for a few years as a simple introduction to the language.
6. Coronado Enterprises Ada 95 Tutorial: shareware edition[10], commercial edition[11].

1 http://en.wikibooks.org/wiki/Ada%20Programming
2 Chapter 23 on page 219
3 http://en.wikibooks.org/wiki/Ada%20Programming%2FAda%202012
4 http://www.dwheeler.com/lovelace/
5 http://www.adatutor.com/
6 http://zhu-qy.blogspot.com.es/2012/08/adatutor.html
7 http://www.adahome.com/Ammo/Cplpl2Ada.html
8 http://home.agh.edu.pl/~jpi/download/ada/guide-c2ada.pdf
9 http://goanna.cs.rmit.edu.au/~dale/ada/aln.html
10 http://www.infres.enst.fr/~pautet/Ada95/a95list.htm
11 http://www.coronadoenterprises.com/tutorials/ada95/index.html

59 Web 2.0

Here is a list of Web 2.0[1] resources about Ada:

59.0.4 News & Blogs

- reddit.com — Ada[2] [RSS[3]], social news website on which users can post links to content on the web
- Ada Gems[4] [RSS[5]], programming tips and articles about specific language features
- Planet Ada[6] [RSS[7]], an aggregate feed of mostly Ada-related blogs.
- Ada Programming blog[8] [RSS[9]], by Martin Krischik and other authors
- Kickin' the Darkness[10] [RSS[11]], by Marc A. Criley
- Archeia[12] [RSS[13]], by Lucretia
- Pragmatic Revelations[14] [RSS[15]]

59.0.5 Forums & developer rings

- Stack Overflow — Ada questions[16]
- Linked In — Ada developers group[17] (free register needed)
- Tek-Tips — Ada Forum[18]

1 http://en.wikipedia.org/wiki/Web%202.0
2 http://www.reddit.com/r/ada/
3 http://www.reddit.com/r/ada/.rss
4 http://www.adacore.com/category/developers-center/gems/
5 http://www.adacore.com/rss/gems
6 http://planet.ada.cx/
7 http://planet.ada.cx/rss20.xml
8 http://ada-programming.blogspot.com/
9 http://ada-programming.blogspot.com/feeds/posts/default
10 http://blog.kickin-the-darkness.com/search/label/Ada
11 http://blog.kickin-the-darkness.com/feeds/posts/default
12 http://www.archeia.com/
13 http://www.archeia.com/rss_feed.html
14 http://adrianhoe.com/adrianhoe/category/software_development/ada/
15 http://adrianhoe.com/adrianhoe/feed/rss/
16 http://stackoverflow.com/questions/tagged/ada
17 http://www.linkedin.com/groups?gid=114211
18 http://www.tek-tips.com/threadminder.cfm?pid=199

59.0.6 General Info

- SlideShare[19], presentations about Ada[20]. See also: Ada programming[21], Ada 95[22], Ada 2005[23], Ada 2012[24] tag pages.
- Ohloh[25], a directory of Open Source projects. Its main features are source code analysis[26] of public repositories and public reviews of projects
- Ada Commons[27], wiki for Ada developers
- Ada@Krischik[28], Ada homepage of Martin Krischik
- WikiCFP — Calls For Papers on Ada[29] [RSS[30]]
- AdaCore channel on youtube.com[31], Ada related videos.

59.0.7 Wikimedia projects

- **Wikipedia articles** (Ada category[32]):
  - Ada[33]
  - Jean Ichbiah[34]
  - Beaujolais effect[35]
  - ISO 8652[36]
  - Ada Semantic Interface Specification[37]
  - ...
- **Wiktionary entries**:
  - ACATS[38]
  - Ada[39]
  - ASIS[40]
- **Wikisource documents**:
  - Steelman language requirements[41]
  - Stoneman requirements[42]
- **Wikibooks tutorials**:

19 http://www.slideshare.net/
20 http://www.slideshare.net/group/ada-programming/slideshows
21 http://www.slideshare.net/tag/ada-programming
22 http://www.slideshare.net/tag/ada-95
23 http://www.slideshare.net/tag/ada-2005
24 http://www.slideshare.net/tag/ada-2012
25 http://www.ohloh.net/tags/ada
26 http://www.ohloh.net/languages/21
27 http://commons.ada.cx
28 http://ada.krischik.com
29 http://www.wikicfp.com/cfp/call?conference=ada
30 http://www.wikicfp.com/cfp/rss?cat=ada
31 http://www.youtube.com/user/AdaCore05
32 http://en.wikipedia.org/wiki/Category%3AAda%20programming%20language
33 http://en.wikipedia.org/wiki/Ada%20%28programming%20language%29
34 http://en.wikipedia.org/wiki/Jean%20Ichbiah
35 http://en.wikipedia.org/wiki/Beaujolais%20effect
36 http://en.wikipedia.org/wiki/ISO%208652
37 http://en.wikipedia.org/wiki/Ada%20Semantic%20Interface%20Specification
38 http://en.wiktionary.org/wiki/ACATS
39 http://en.wiktionary.org/wiki/Ada
40 http://en.wiktionary.org/wiki/ASIS
41 http://en.wikisource.org/wiki/Steelman%20language%20requirements
42 http://en.wikisource.org/wiki/Stoneman%20requirements

- *Programación en Ada*[43], in Spanish
- *Programmation Ada*[44], in French
- *Ada*[45], in Italian

- **Wikiquote**:
  - Programming languages — Ada[46]
- **Wikiversity**:
  - Ada course[47] (you can enroll!)

59.0.8 Source code

- Examples *Ada Programming* wikibook[48]
- Rosetta Code — Ada Category[49], programming examples in multiple languages
- literateprograms.org — Ada Category[50], examples of literate programming[51] in multiple languages

59.0.9 Projects

- AdaCL[52]
- The Ada 95 Booch Components[53]
- The GNU Ada Compiler[54]
- ASIS[55]
- GLADE[56]
- Florist[57]
- GNAT — GCC Wiki[58]
- RTEMSAda[59]
- AVR-Ada[60] - Ada compiler for Atmel microcontrollers (Arduinos)

Web 2.0[61]

43 http://en.wikibooks.org/wiki/%3Aes%3AProgramaci%F3n_en_Ada
44 http://en.wikibooks.org/wiki/%3Afr%3AProgrammation_Ada
45 http://en.wikibooks.org/wiki/%3Ait%3AAda
46 http://en.wikiquote.org/wiki/Programming_languages#Ada
47 http://en.wikiversity.org/wiki/Ada
48 http://wikibook-ada.sourceforge.net
49 http://www.rosettacode.org/wiki/Ada
50 http://en.literateprograms.org/Category:Programming_language:Ada
51 http://en.wikipedia.org/wiki/literate%20programming
52 http://adacl.sourceforge.net/index.php
53 http://booch95.sourceforge.net/pmwiki.php
54 http://gnuada.sourceforge.net
55 http://gnat-asis.sourceforge.net
56 http://gnat-glade.sourceforge.net
57 http://gnat-florist.sourceforge.net
58 http://gcc.gnu.org/wiki/GNAT
59 http://www.rtems.com/wiki/index.php/RTEMSAda
60 http://avr-ada.sourceforge.net/
61 http://en.wikibooks.org/wiki/Category%3AAda%20Programming

60 Contributors

Edits	User
1	A10112[1]
1	ALK[2]
1	AdRiley[3]
64	Adrignola[4]
1	Alan.poindexter[5]
2	Alisonken1[6]
1	Ammon[7]
1	Andreas Ipp[8]
1	Aramael[9]
1	Arny[10]
1	Arthurvogel[11]
7	Avicennasis[12]
1	Benjstarratt[13]
14	Carsrac[14]
56	CarsracBot[15]
1	Chouclac[16]
2	Cspurrier[17]
44	Darklama[18]
3	David Hoos[19]
3	Derbeth[20]
1	Dhenry[21]

1 http://en.wikibooks.org/w/index.php?title=User:A10112
2 http://en.wikibooks.org/w/index.php?title=User:ALK
3 http://en.wikibooks.org/w/index.php?title=User:AdRiley
4 http://en.wikibooks.org/w/index.php?title=User:Adrignola
5 http://en.wikibooks.org/w/index.php?title=User:Alan.poindexter
6 http://en.wikibooks.org/w/index.php?title=User:Alisonken1
7 http://en.wikibooks.org/w/index.php?title=User:Ammon
8 http://en.wikibooks.org/w/index.php?title=User:Andreas_Ipp
9 http://en.wikibooks.org/w/index.php?title=User:Aramael
10 http://en.wikibooks.org/w/index.php?title=User:Arny
11 http://en.wikibooks.org/w/index.php?title=User:Arthurvogel
12 http://en.wikibooks.org/w/index.php?title=User:Avicennasis
13 http://en.wikibooks.org/w/index.php?title=User:Benjstarratt
14 http://en.wikibooks.org/w/index.php?title=User:Carsrac
15 http://en.wikibooks.org/w/index.php?title=User:CarsracBot
16 http://en.wikibooks.org/w/index.php?title=User:Chouclac
17 http://en.wikibooks.org/w/index.php?title=User:Cspurrier
18 http://en.wikibooks.org/w/index.php?title=User:Darklama
19 http://en.wikibooks.org/w/index.php?title=User:David_Hoos
20 http://en.wikibooks.org/w/index.php?title=User:Derbeth
21 http://en.wikibooks.org/w/index.php?title=User:Dhenry

12 Dirk Hünniger[22]
2 Dmitry-kazakov[23]
1 Doug[24]
2 DougP[25]
17 Dragontamer[26]
1 Fraterm[27]
1 Friess[28]
1 Frikk[29]
1 Frodet[30]
2 Geocachernemesis[31]
97 GeorgBauhaus[32]
8 Godunko[33]
1 Hagindaz[34]
1 Herbythyme[35]
2 JMatthews[36]
3 James Dennett[37]
1 Jclee[38]
1 Jcreem[39]
1 Jeffinous[40]
5 Jesselang[41]
34 Jguk[42]
1 Jlaire[43]
2 Jlenthe[44]
4 Jomegat[45]
1 Kayau[46]

22 http://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger
23 http://en.wikibooks.org/w/index.php?title=User:Dmitry-kazakov
24 http://en.wikibooks.org/w/index.php?title=User:Doug
25 http://en.wikibooks.org/w/index.php?title=User:DougP
26 http://en.wikibooks.org/w/index.php?title=User:Dragontamer
27 http://en.wikibooks.org/w/index.php?title=User:Fraterm
28 http://en.wikibooks.org/w/index.php?title=User:Friess
29 http://en.wikibooks.org/w/index.php?title=User:Frikk
30 http://en.wikibooks.org/w/index.php?title=User:Frodet
31 http://en.wikibooks.org/w/index.php?title=User:Geocachernemesis
32 http://en.wikibooks.org/w/index.php?title=User:GeorgBauhaus
33 http://en.wikibooks.org/w/index.php?title=User:Godunko
34 http://en.wikibooks.org/w/index.php?title=User:Hagindaz
35 http://en.wikibooks.org/w/index.php?title=User:Herbythyme
36 http://en.wikibooks.org/w/index.php?title=User:JMatthews
37 http://en.wikibooks.org/w/index.php?title=User:James_Dennett
38 http://en.wikibooks.org/w/index.php?title=User:Jclee
39 http://en.wikibooks.org/w/index.php?title=User:Jcreem
40 http://en.wikibooks.org/w/index.php?title=User:Jeffinous
41 http://en.wikibooks.org/w/index.php?title=User:Jesselang
42 http://en.wikibooks.org/w/index.php?title=User:Jguk
43 http://en.wikibooks.org/w/index.php?title=User:Jlaire
44 http://en.wikibooks.org/w/index.php?title=User:Jlenthe
45 http://en.wikibooks.org/w/index.php?title=User:Jomegat
46 http://en.wikibooks.org/w/index.php?title=User:Kayau

1102	Krischik[47]
1	Kwhitefoot[48]
16	Larry Luther[49]
3	LesmanaZimmer[50]
1	Lincher[51]
1	Lodacom[52]
118	Ludovic Brenta[53]
1	Maciej Sobczak[54]
2	Mahanga[55]
463	ManuelGR[56]
4	Mike.lifeguard[57]
2	Moskvax[58]
5	Nikai[59]
2	Okellogg[60]
6	Oleszkie[61]
8	Panic2k4[62]
1	Panzon[63]
1	Parallelized[64]
2	Paxton[65]
1	Per.sandberg[66]
1	QuiteUnusual[67]
1	RamaccoloBot[68]
2	Randhol[69]
1	Recent Runes[70]
1	Red4tribe[71]

47 http://en.wikibooks.org/w/index.php?title=User:Krischik
48 http://en.wikibooks.org/w/index.php?title=User:Kwhitefoot
49 http://en.wikibooks.org/w/index.php?title=User:Larry_Luther
50 http://en.wikibooks.org/w/index.php?title=User:LesmanaZimmer
51 http://en.wikibooks.org/w/index.php?title=User:Lincher
52 http://en.wikibooks.org/w/index.php?title=User:Lodacom
53 http://en.wikibooks.org/w/index.php?title=User:Ludovic_Brenta
54 http://en.wikibooks.org/w/index.php?title=User:Maciej_Sobczak
55 http://en.wikibooks.org/w/index.php?title=User:Mahanga
56 http://en.wikibooks.org/w/index.php?title=User:ManuelGR
57 http://en.wikibooks.org/w/index.php?title=User:Mike.lifeguard
58 http://en.wikibooks.org/w/index.php?title=User:Moskvax
59 http://en.wikibooks.org/w/index.php?title=User:Nikai
60 http://en.wikibooks.org/w/index.php?title=User:Okellogg
61 http://en.wikibooks.org/w/index.php?title=User:Oleszkie
62 http://en.wikibooks.org/w/index.php?title=User:Panic2k4
63 http://en.wikibooks.org/w/index.php?title=User:Panzon
64 http://en.wikibooks.org/w/index.php?title=User:Parallelized
65 http://en.wikibooks.org/w/index.php?title=User:Paxton
66 http://en.wikibooks.org/w/index.php?title=User:Per.sandberg
67 http://en.wikibooks.org/w/index.php?title=User:QuiteUnusual
68 http://en.wikibooks.org/w/index.php?title=User:RamaccoloBot
69 http://en.wikibooks.org/w/index.php?title=User:Randhol
70 http://en.wikibooks.org/w/index.php?title=User:Recent_Runes
71 http://en.wikibooks.org/w/index.php?title=User:Red4tribe

5 Robert Horning[72]
1 Rosen[73]
1 Rursus[74]
3 SQL[75]
2 Sam[76]
5 Sjw[77]
1 Sparre[78]
5 Spongebob88[79]
1 Stephen leake[80]
711 Suruena[81]
2 Swhalen[82]
2 The bellman[83]
3 Thenub314[84]
3 Tkoskine[85]
1 Tobias Bergemann[86]
1 Van der Hoorn[87]
3 Venullian[88]
5 VillemtheVillain![89]
1 Vito Genovese[90]
1 Warinthepocket[91]
3 WhirlWind[92]
1 Wikibob[93]
1 Xania[94]
1 robot[95]

72 http://en.wikibooks.org/w/index.php?title=User:Robert_Horning
73 http://en.wikibooks.org/w/index.php?title=User:Rosen
74 http://en.wikibooks.org/w/index.php?title=User:Rursus
75 http://en.wikibooks.org/w/index.php?title=User:SQL
76 http://en.wikibooks.org/w/index.php?title=User:Sam
77 http://en.wikibooks.org/w/index.php?title=User:Sjw
78 http://en.wikibooks.org/w/index.php?title=User:Sparre
79 http://en.wikibooks.org/w/index.php?title=User:Spongebob88
80 http://en.wikibooks.org/w/index.php?title=User:Stephen_leake
81 http://en.wikibooks.org/w/index.php?title=User:Suruena
82 http://en.wikibooks.org/w/index.php?title=User:Swhalen
83 http://en.wikibooks.org/w/index.php?title=User:The_bellman
84 http://en.wikibooks.org/w/index.php?title=User:Thenub314
85 http://en.wikibooks.org/w/index.php?title=User:Tkoskine
86 http://en.wikibooks.org/w/index.php?title=User:Tobias_Bergemann
87 http://en.wikibooks.org/w/index.php?title=User:Van_der_Hoorn
88 http://en.wikibooks.org/w/index.php?title=User:Venullian
89 http://en.wikibooks.org/w/index.php?title=User:VillemtheVillain%21
90 http://en.wikibooks.org/w/index.php?title=User:Vito_Genovese
91 http://en.wikibooks.org/w/index.php?title=User:Warinthepocket
92 http://en.wikibooks.org/w/index.php?title=User:WhirlWind
93 http://en.wikibooks.org/w/index.php?title=User:Wikibob
94 http://en.wikibooks.org/w/index.php?title=User:Xania
95 http://en.wikibooks.org/w/index.php?title=User:%E3%82%BF%E3%83%81%E3%82%B3%E3%83%9E_robot

List of Figures

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. http://artlibre.org/licence/lal/de
- CFR: Copyright free use.
- EPL: Eclipse Public License. http://www.eclipse.org/org/documents/epl-v10.php

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[96]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

96 Chapter 61 on page 399

1	ManuelGR[97]	GFDL

97 http://en.wikibooks.org/wiki/User%3AManuelGR

61 Licenses

61.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007
Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble
The GNU General Public License is a free, copyleft license for software and other kinds of works.
The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.
When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.
To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.
For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.
Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.
For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.
Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.
Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.
The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.
"This License" refers to version 3 of the GNU General Public License.
"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.
"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.
To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.
A "covered work" means either the unmodified Program or a work based on the Program.
To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.
To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.
An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.
The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.
A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.
The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.
The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.
The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.
The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.
All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.
You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.
Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.
No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.
When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.
You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program. You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.
You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:
* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.
A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.
You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:
* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.
A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.
A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.
"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.
If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).
The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.
Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.
"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.
When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.
Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:
* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.
All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.
If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.
Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.
You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.
You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.
Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.
An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.
You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.
A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".
A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.
Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.
In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To

"grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.
If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.
If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.
A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.
Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.
If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.
Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.
The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.
If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.
Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.
THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.
IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.
If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.
END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs
If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the f notice is found.
<one line to give the program's name and a br idea of what it does.> Copyright (C) <yea <name of author>
This program is free software: you can redistribu it and/or modify it under the terms of the GN General Public License as published by the F Software Foundation, either version 3 of the cense, or (at your option) any later version.
This program is distributed in the hope th it will be useful, but WITHOUT ANY WA RANTY; without even the implied warranty MERCHANTABILITY or FITNESS FOR A PA TICULAR PURPOSE. See the GNU General Pub License for more details.
You should have received a copy of the GNU G eral Public License along with this program. If n see <http://www.gnu.org/licenses/>.
Also add information on how to contact you by el tronic and paper mail.
If the program does terminal interaction, make output a short notice like this when it starts in interactive mode:
<program> Copyright (C) <year> <name of thor> This program comes with ABSOLUTE NO WARRANTY; for details type 'show w'. This free software, and you are welcome to redistribute under certain conditions; type 'show c' for detai The hypothetical commands 'show w' and 'show should show the appropriate parts of the Gene Public License. Of course, your program's co mands might be different; for a GUI interface, y would use an "about box".
You should also get your employer (if you wo as a programmer) or school, if any, to sign "copyright disclaimer" for the program, if n essary. For more information on this, a how to apply and follow the GNU GPL, s <http://www.gnu.org/licenses/>.
The GNU General Public License does not perm incorporating your program into proprietary p grams. If your program is a subroutine library, y may consider it more useful to permit linking p prietary applications with the library. If this what you want to do, use the GNU Lesser Gene Public License instead of this License. But fi please read <http://www.gnu.org/philosophy/w not-lgpl.html>.

61.2 GNU Free Documentation License

Version 1.3, 3 November 2008
Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE
The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS
This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.
A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.
A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.
The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.
The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.
A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".
Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.
The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.
The "publisher" means any person or entity that distributes copies of the Document to the public.
A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.
The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING
You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.
You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY
If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS
You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:
* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
* K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.
If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.
You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the a thoritative definition of a standard.
You may add a passage of up to five words as Front-Cover Text, and a passage of up to 25 wo as a Back-Cover Text, to the end of the list of Co Texts in the Modified Version. Only one passage Front-Cover Text and one of Back-Cover Text m be added by (or through arrangements made b any one entity. If the Document already inclu a cover text for the same cover, previously add by you or by arrangement made by the same ent you are acting on behalf of, you may not add other; but you may replace the old one, on expli permission from the previous publisher that add the old one.
The author(s) and publisher(s) of the Document not by this License give permission to use th names for publicity for or to assert or imply dorsement of any Modified Version. 5. COMBI ING DOCUMENTS
You may combine the Document with other do ments released under this License, under the ter defined in section 4 above for modified versio provided that you include in the combination of the Invariant Sections of all of the original d uments, unmodified, and list them all as Invari Sections of your combined work in its license tice, and that you preserve all their Warranty D claimers.
The combined work need only contain one copy this License, and multiple identical Invariant S tions may be replaced with a single copy. If th are multiple Invariant Sections with the same na but different contents, make the title of each su section unique by adding at the end of it, in par theses, the name of the original author or publish of that section if known, or else a unique numb Make the same adjustment to the section titles the list of Invariant Sections in the license not of the combined work.
In the combination, you must combine any sectio Entitled "History" in the various original do ments, forming one section Entitled "History"; li wise combine any sections Entitled "Acknowled ments", and any sections Entitled "Dedication You must delete all sections Entitled "Endor ments". 6. COLLECTIONS OF DOCUMENTS
You may make a collection consisting of the Do ment and other documents released under this cense, and replace the individual copies of this cense in the various documents with a single co that is included in the collection, provided that y follow the rules of this License for verbatim copyi of each of the documents in all other respects.
You may extract a single document from such a c lection, and distribute it individually under this cense, provided you insert a copy of this Licen into the extracted document, and follow this cense in all other respects regarding verbatim co ing of that document. 7. AGGREGATION WI INDEPENDENT WORKS
A compilation of the Document or its derivati with other separate and independent documents works, in or on a volume of a storage or distributi medium, is called an "aggregate" if the copyright sulting from the compilation is not used to limit t legal rights of the compilation's users beyond wh the individual works permit. When the Docume is included in an aggregate, this License does n apply to the other works in the aggregate which a not themselves derivative works of the Documen If the Cover Text requirement of section 3 is ap cable to these copies of the Document, then if t Document is less than one half of the entire agg gate, the Document's Cover Texts may be pla on covers that bracket the Document within t aggregate, or the electronic equivalent of cov if the Document is in electronic form. Otherw they must appear on printed covers that brack the whole aggregate. 8. TRANSLATION
Translation is considered a kind of modification, you may distribute translations of the Docume under the terms of section 4. Replacing Invaria Sections with translations requires special perm sion from their copyright holders, but you may clude translations of some or all Invariant Sectio in addition to the original versions of these Inva ant Sections. You may include a translation of t License, and all the license notices in the Docume and any Warranty Disclaimers, provided that y

also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.
9. TERMINATION
You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.
However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE
The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.
11. RELICENSING
"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.
"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.
"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.
An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.
The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents
To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:
Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:
with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.
If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

61.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE
Version 3, 29 June 2007
Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.
0. Additional Definitions.
As used herein, "this License" refers to version 3 of the GNU Lesser General Public License, and the "GNU GPL" refers to version 3 of the GNU General Public License.
"The Library" refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.
An "Application" is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.
A "Combined Work" is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the "Linked Version".
The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.
The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.
You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.
If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:
* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.
3. Object Code Incorporating Material from Library Header Files.
The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:
* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.
4. Combined Works.
You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:
* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)
5. Combined Libraries.
You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:
* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.
6. Revised Versions of the GNU Lesser General Public License.
The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.
If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.