Wikibook's Ada Programming

Martin Krischik
Manuel Gmez
Santiago Uruea
C.K.W. Grein
Bill Findlay
B. Seidel
Simon Wright
Allen Lew
John Oleszkiewicz
Nicolas Kaiser
Larry Luther
Georg Bauhaus
Samuel Tardieu
Ludovic Brenta
Ed Falis
Pascal Obry
And many anonymous Wikibook Contributors

December 25, 2006

Contents

Preface

Welcome to the Ada Programming tutorial at Wikibooks. This is the first Ada tutorial covering the imminent Ada 2005 standard. If you are a beginner you will learn the future standard - if you are a seasoned Ada user you can see what's new.

Current Development Stage for Ada Programming is "(Jul 27, 2005)". At this date, there are more than 200 pages in this book, which makes Ada Programming the largest of the programming wikibooks.

But still there is always room for improvement do help us to expand Ada Programming. Even beginners will find areas to participate.

About Ada

Ada is a programming language named after Augusta Ada King, Countess of Lovelace, which is suitable for all development needs.

Ada has built-in features that directly support structured, object-oriented, generic, distributed and concurrent programming.

Ada is a good choice for Rapid Application Development, Extreme Programming (XP), and Free Software development.

Programming in the large

Ada puts unique emphasis on, and provides strong support for, good software engineering practices that scale well to very large software systems (millions of lines of code, and very large development teams). The following language features are particularly relevant in this respect:

- An extremely strong, static and safe *type system*, which allows the programmer to construct powerful abstractions that reflect the real world, and allows the compiler to detect many logic errors before they become bugs.
- *Modularity*, whereby the compiler directly manages the construction of very large software systems from sources.
- *Information hiding*; the language separates interfaces from implementation, and provides fine-grained control over visibility.
- *Readability*, which helps programmers review and verify code. Ada favours the reader of the program over the writer, because a program is written once but read many times. For example, the syntax bans all ambiguous constructs, so there are no surprises, in accordance with

the Tao of Programming's Law of Least Astonishment [1]. (Some Ada programmers are reluctant to talk about *source code* which is often cryptic; they prefer *program text* which is close to English prose.)

- *Portability*: the language definition allows compilers to differ only in a few controlled ways, and otherwise defines the semantics of programs very precisely; as a result, Ada source text is very portable across compilers and across target hardware platforms. Most often, the program can be recompiled without any changes (see [1]).

- *Standardisation*: standards have been a goal and a prominent feature ever since the design of the language in the late 1970's. The first standard was published in 1980, just 3 years after design commenced; and a new revision of the standard is being finalised in 2006. Ada compilers all support the exact same language; there are no dialects.

Consequences of these qualities are superior reliability, reusability and maintainability. For example, compared to programs written in C, programs written in Ada 83 contain ten times fewer bugs, and cost half as much to develop in the first place (see [2]). Ada shines even more in software maintenance, which often accounts for 80% of the total cost of development. With support for object-oriented programming, Ada95 brings even more cost benefits, although no serious study comparable to Zeigler's has been published.

Programming in the small

In addition to its support for good software engineering practices, which are applicable to general-purpose programming, Ada has powerful specialised features supporting low-level programming for real-time, safety-critical and embedded systems. Such features include, among others, machine code insertions, address arithmetic, low-level access to memory, control over bitwise representation of data, bit manipulations, and a well-defined, statically provable concurrent computing model called the Ravenscar Profile.

Other features include restrictions (it is possible to restrict which language features are accepted in a program) and features that help review and certify the object code generated by the compiler.

[1]http://www.brandwand.com/tao/tao_4.html

Several vendors provide Ada compilers accompanied by minimal run-time kernels suitable for use in certified, life-critical applications. It is also possible to write Ada programs which require no run-time kernel at all.

It should come as no surprise that Ada is heavily used in the aerospace, defence, medical, railroad, and nuclear industries.

The Language Reference Manual

The Ada Reference Manual (RM) — full name Ada Reference Manual, ISO/IEC 8652:1995(E) – is the official language definition. If you have a question no one can answer for you, you will find something in the RM (albeit often a bit cryptic for non-language-lawyers). For this reason, all complete (not draft) pages in Ada Programming contain links into the appropriate pages in the RM.

- You can browse the complete RM for Ada 95 in one of the following sites:
  - `http://www.adaic.org/standards/95lrm/html/RM-TTL.html`
  - `http://www.adapower.com/rm95.php`
  - `http://www.adaic.org/standards/95lrm/html/RM-TTL.html`
  - `http://www.adapower.com/rm95.php`
- There are two documents related with the RM:
  - The Annotated Reference Manual[2], an extended version of the RM aimed at compiler writers or other persons who want to know the fine details of the language.
  - The Reference Manual Rationale[3], an explanation of the features of the language.
- The draft [4] for the upcoming Ada 2005 can be found here. Beware: it's not finished yet.

[2]http://www.adaic.org/standards/95aarm/html/AA-TTL.html
[3]http://www.adaic.org/standards/95rat/RAThtml/rat95-contents.html
[4]http://www.adaic.org/standards/ada05.html

Ada Conformity Assessment Test Suite

Unlike other programming languages, Ada compilers are officially tested, and only those which pass this test are accepted, for military and commercial work. This means that all Ada compilers behave (almost) the same, so you do not have to learn any dialects. But because the Ada standard allows the compiler writers to include some additions, you could learn a cool new feature only to find out that your favorite compiler does not support it...

Part I

Getting started

Chapter 1

Basic Ada

“Hello, world!” programs

“Hello, world!”

A common example of a language’s syntax is the Hello world program. Here a straight-forward Ada Implementation:

```
with Ada.Text_IO;

procedure Hello_World_1 is
begin
   Ada.Text_IO.Put_Line ("Hello World!");
end Hello_World_1;
```

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 (i.e. all types and the operations on them) within Ada.Text_IO visible by selection. In Ada, packages can be used as toolboxes, providing a collection of tools and types 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;

   -- more stuff

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

   -- more stuff

   procedure Put_Line (Item : String);

   -- more stuff

end Ada.Text_IO;
```

“Hello, world!” with renames

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

```
with Ada.Text_IO;

procedure Hello_World_2 is
   package IO renames Ada.Text_IO;
begin
   IO.Put_Line ("Hello World!");
   IO.New_Line;
   IO.Put_Line ("I am an Ada programm with package rename.");
end Hello_World_2;
```

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.

"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).

```ada
with Ada.Text_IO;

procedure Hello_World_3 is
   use Ada.Text_IO;
begin
   Put_Line ("Hello World!");
   New_Line;
   Put_Line ("I am an Ada programm with package use.");
end Hello_World_3;
```

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

FAQ: Why is "Hello, world!" sooo 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 not frighten away potential new users by providing a system which does not work "out of the box".

The GNAT project files, which you can download 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
```

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.

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.

Bibliography

[1] Dirk Craeynest Gaetan Allaert and Philippe Waroquiers. European Air Traffic Flow Management:Porting a Large Application to GNU/Linux. 2003.

[2] Stephen F. Zeigler. Comparing Development Costs of C and Ada. 1995.