**Download Ada compiler**

Use the following web page to download GNAT compiler:

<http://libre.adacore.com/download/configurations>

Select the platform-form specific bundle of GNAT GPL 2016.

For Windows, this is a ZIP archive.

See options for Mac OSX

On Linux, use a package manager to download GNAT.

On Windows, extract the content of the ZIP archive.

Run the executable file, this will install the compiler and support files.

Open a command window.

Change directory to the one where your Ada programs are stored.

Compile, bind, and link (see below)

**Example:**

On a command window (on my laptop):

I stored sample Ada source programs on the directory C:\ada\_progs

C:\ada\_progs>dir

Volume in drive C is OS

Volume Serial Number is 9EC6-ED87

Directory of C:\ada\_progs

04/05/2017 09:01 AM <DIR> .

04/05/2017 09:01 AM <DIR> ..

04/04/2017 05:59 PM 106 hello.adb

11/03/2016 12:44 PM 1,604 tokens.ads

2 File(s) 1,710 bytes

2 Dir(s) 735,401,541,632 bytes free

C:\ada\_progs>gcc -c hello.adb

C:\ada\_progs>dir

Volume in drive C is OS

Volume Serial Number is 9EC6-ED87

Directory of C:\ada\_progs

04/05/2017 09:02 AM <DIR> .

04/05/2017 09:02 AM <DIR> ..

04/04/2017 05:59 PM 106 hello.adb

04/05/2017 09:02 AM 1,678 hello.ali

04/05/2017 09:02 AM 855 hello.o

11/03/2016 12:44 PM 1,604 tokens.ads

4 File(s) 4,243 bytes

2 Dir(s) 735,401,447,424 bytes free

C:\ada\_progs>gnatbind hello

C:\ada\_progs>gnatlink hello

C:\ada\_progs>dir

Volume in drive C is OS

Volume Serial Number is 9EC6-ED87

Directory of C:\ada\_progs

04/05/2017 09:03 AM <DIR> .

04/05/2017 09:03 AM <DIR> ..

04/04/2017 05:59 PM 106 hello.adb

04/05/2017 09:02 AM 1,678 hello.ali

04/05/2017 09:03 AM 912,856 hello.exe

04/05/2017 09:02 AM 855 hello.o

11/03/2016 12:44 PM 1,604 tokens.ads

5 File(s) 917,099 bytes

2 Dir(s) 735,399,944,192 bytes free

C:\ada\_progs>hello

Hello WORLD!

C:\ada\_progs>

**2.1 Running GNAT**

Three steps are needed to create an executable file from an Ada source file:

* \* The source file(s) must be compiled.
* \* The file(s) must be bound using the GNAT binder.
* \* All appropriate object files must be linked to produce an executable.

All three steps are most commonly handled by using the `gnatmake' utility program that, given the name of the main program, automatically performs the necessary compilation, binding and linking steps.

**2.2 Running a Simple Ada Program**

Any text editor may be used to prepare an Ada program. (If Emacs is used, the optional Ada mode may be helpful in laying out the program.) The program text is a normal text file. We will assume in our initial example that you have used your editor to prepare the following standard format text file:

with Ada.Text\_IO; use Ada.Text\_IO;

procedure Hello is

begin

Put\_Line ("Hello WORLD!");

end Hello;

This file should be named hello.adb. With the normal default file naming conventions, GNAT requires that each file contain a single compilation unit whose file name is the unit name, with periods replaced by hyphens; the extension is ads for a spec and adb for a body. You can override this default file naming convention by use of the special pragma *Source\_File\_Name* (for further information please see [Using Other File Names](https://gcc.gnu.org/onlinedocs/gnat_ugn/Using-Other-File-Names.html#g_t35)). Alternatively, if you want to rename your files according to this default convention, which is probably more convenient if you will be using GNAT for all your compilations, then the *gnatchop* utility can be used to generate correctly-named source files (see [Renaming Files with gnatchop](https://gcc.gnu.org/onlinedocs/gnat_ugn/Renaming-Files-with-gnatchop.html#g_t36)).

You can compile the program using the following command (*$* is used as the command prompt in the examples in this document):

$ gcc -c hello.adb

`gcc' is the command used to run the compiler. This compiler is capable of compiling programs in several languages, including Ada and C. It assumes that you have given it an Ada program if the file extension is either .ads or .adb, and it will then call the GNAT compiler to compile the specified file.

The -c switch is required. It tells `gcc' to only do a compilation. (For C programs, `gcc' can also do linking, but this capability is not used directly for Ada programs, so the -c switch must always be present.)

This compile command generates a file hello.o, which is the object file corresponding to your Ada program. It also generates an ’Ada Library Information’ file hello.ali, which contains additional information used to check that an Ada program is consistent. To build an executable file, use *gnatbind* to bind the program and `gnatlink' to link it. The argument to both *gnatbind* and `gnatlink' is the name of the ALI file, but the default extension of .ali can be omitted. This means that in the most common case, the argument is simply the name of the main program:

$ gnatbind hello

$ gnatlink hello

A simpler method of carrying out these steps is to use `gnatmake', a master program that invokes all the required compilation, binding and linking tools in the correct order. In particular, `gnatmake' automatically recompiles any sources that have been modified since they were last compiled, or sources that depend on such modified sources, so that ’version skew’ is avoided.

$ gnatmake hello.adb

The result is an executable program called hello, which can be run by entering:

$ hello

assuming that the current directory is on the search path for executable programs.

and, if all has gone well, you will see:

Hello WORLD!

appear in response to this command.

**2.3 Running a Program with Multiple Units**

Consider a slightly more complicated example that has three files: a main program, and the spec and body of a package:

package Greetings is

procedure Hello;

procedure Goodbye;

end Greetings;

with Ada.Text\_IO; use Ada.Text\_IO;

package body Greetings is

procedure Hello is

begin

Put\_Line ("Hello WORLD!");

end Hello;

procedure Goodbye is

begin

Put\_Line ("Goodbye WORLD!");

end Goodbye;

end Greetings;

with Greetings;

procedure Gmain is

begin

Greetings.Hello;

Greetings.Goodbye;

end Gmain;

Following the one-unit-per-file rule, place this program in the following three separate files:

`greetings.ads'

spec of package *Greetings*

`greetings.adb'

body of package *Greetings*

`gmain.adb'

body of main program

To build an executable version of this program, we could use four separate steps to compile, bind, and link the program, as follows:

$ gcc -c gmain.adb

$ gcc -c greetings.adb

$ gnatbind gmain

$ gnatlink gmain

Note that there is no required order of compilation when using GNAT. In particular it is perfectly fine to compile the main program first. Also, it is not necessary to compile package specs in the case where there is an accompanying body; you only need to compile the body. If you want to submit these files to the compiler for semantic checking and not code generation, then use the -gnatc switch:

$ gcc -c greetings.ads -gnatc

Although the compilation can be done in separate steps as in the above example, in practice it is almost always more convenient to use the `gnatmake' tool. All you need to know in this case is the name of the main program’s source file. The effect of the above four commands can be achieved with a single one:

$ gnatmake gmain.adb

In the next section we discuss the advantages of using `gnatmake' in more detail.

**2.4 Using the `gnatmake' Utility**

If you work on a program by compiling single components at a time using `gcc', you typically keep track of the units you modify. In order to build a consistent system, you compile not only these units, but also any units that depend on the units you have modified. For example, in the preceding case, if you edit gmain.adb, you only need to recompile that file. But if you edit greetings.ads, you must recompile both greetings.adb and gmain.adb, because both files contain units that depend on greetings.ads.

`gnatbind' will warn you if you forget one of these compilation steps, so that it is impossible to generate an inconsistent program as a result of forgetting to do a compilation. Nevertheless it is tedious and error-prone to keep track of dependencies among units. One approach to handle the dependency-bookkeeping is to use a makefile. However, makefiles present maintenance problems of their own: if the dependencies change as you change the program, you must make sure that the makefile is kept up-to-date manually, which is also an error-prone process.

The `gnatmake' utility takes care of these details automatically. Invoke it using either one of the following forms:

$ gnatmake gmain.adb

$ gnatmake gmain

The argument is the name of the file containing the main program; you may omit the extension. `gnatmake' examines the environment, automatically recompiles any files that need recompiling, and binds and links the resulting set of object files, generating the executable file, gmain. In a large program, it can be extremely helpful to use `gnatmake', because working out by hand what needs to be recompiled can be difficult.

Note that `gnatmake' takes into account all the Ada rules that establish dependencies among units. These include dependencies that result from inlining subprogram bodies, and from generic instantiation. Unlike some other Ada make tools, `gnatmake' does not rely on the dependencies that were found by the compiler on a previous compilation, which may possibly be wrong when sources change. `gnatmake' determines the exact set of dependencies from scratch each time it is run.