UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS61B **P. N. Hilfinger**
Spring 1998

Using `gcc`

`Gcc` is a publicly-available optimizing compiler (translator) for C, C++, Ada 95, and Objective C that currently runs under various implementations of Unix (plus VMS as well as OS/2 and perhaps other PC systems) on a variety of processors too numerous to mention. You can find full documentation on-line under Emacs (use `C-h i` and select the "GCC" menu option). You don't need to know much about it for our purposes. This document is a brief summary.

Running the compiler

You can use `gcc` both to compile programs into object modules and to link these object modules together into a single program. It looks at the names of the files you give it to determine what language they are in and what to do with them. Files of the form *name*`.cc` (or *name*`.C`) are assumed to be C++ files and files matching *name*`.o` are assumed to be object (i.e., machine-language) files. For compiling C++ programs, you should use `g++`, which is basically an alias for `gcc` that automatically includes certain libraries that are used in C++, but not C.

To translate a C++ source file, F`.cc`, into a corresponding object file, F`.o`, use the command

`g++ -c` *compile-options* F`.cc`

To link one or more object files—F_1`.o`, F_2`.o`, ...—produced from C++ files into a single executable file called F, use the command.

`g++ -o` F *link-options* F_1`.o` F_2`.o ...` *libraries*

(The *options* and *libraries* clauses are described below.)

You can bunch these two steps—compilation and linking—into one with the following command.

`g++ -o` F *compile-and-link-options* F_1`.cc ...` *other-libraries*

After linking has produced an executable file called F, it becomes, in effect, a new Unix command, which you can run with

`./`F `arguments`

where *arguments* denotes any command-line arguments to the program.

Libraries

A *library* is a collection of object files that has been grouped together into a single file and indexed. When the linking command encounters a library in its list of object files to link, it looks to see if preceding object files contained calls to functions not yet defined that are defined in one of the library's object files. When it finds such a function, it then links in the appropriate object file from the library. One library gets added to the list of libraries automatically, and provides a number of standard functions common to C++ and C.

Libraries are usually designated with an argument of the form `-l`*library-name*. In particular, `-lm` denotes a library containing various mathematical routines (sine, cosine, arctan, square root, etc.) They must be listed *after* the object or source files that contain calls to their functions.

Options

The following compile- and link-options will be of particular interest to us.

-c (Compilation option)
Compile only. Produces `.o` files from source files without doing any linking.

-D*name***=***value* (Compilation option)
In the program being compiled, define *name* as if there were a line

`#define` *name* *value*

at the beginning of the program. The '=*value*' part may be left off, in which case *value* defaults to 1.

-o *file-name* (Link option, usually)
Use *file-name* as the name of the file produced by `g++` (usually, this is an executable file).

-l*library-name* (Link option)
Link in the specified library. See above. (Link option).

-g (Compilation and link option)
Put debugging information for `gdb` into the object or executable file. Should be specified for *both* compilation and linking.

-MM (Compilation option)
Print the header files (other than standard headers) used by each source file in a format acceptable to `make`. Don't produce a `.o` file or an executable.

-pg (Compilation and link option)
Put profiling instructions for generating profiling information for `gprof` into the object or executable file. Should be specified for *both* compilation or linking. *Profiling* is the process of measuring how long various portions of your program take to execute. When you specify `-pg`, the resulting executable program, when run, will produce a file of statistics. A program called

`gprof` will then produce a listing from that file telling how much time was spent executing each function.

-Wall (Compilation option)
Produce warning messages about a number of things that are legal but dubious. I strongly suggest that you *always* specify this and that you treat every warning as an error to be fixed[1].

[1] You will see that we often use add the cryptic option `-fno-builtins`. We do so because of a temporary glitch in our current version of `g++` that otherwise causes it to suppress some warning messages.