COMPILING A SIMPLE C PROGRAM - II

Compilation of a C program

As a first step a file with the extension .c to contain our program should be created using a test editor. Once the program has been saved with a .c extension file (i.e. prg1.c) then the simplest command to initiate a compilation is: First using a text editor one has to create a file with a .c extension and save the required program. As discussed in the previous chapter, on any system that has C compiler, the simplest command which initiates compilation is:

```
cc prg1.c
```

Given that our system has a cc compiler (included in most UNIX and linux distributions). There are many other C compilers around. The cc is the default Sun compiler. The GNU C compiler gcc is also available for many platforms. Windows users may also be familiar with the Borland bcc compiler. All c compilers operate in the same manner, and share a number of compiler options. Refer to the annexes for a list of available compiler options or use the compiler help (i.e. *man* cc for the help of the cc compiler in UNIX/Linux systems).

Through out this course we will refer to the cc compiler. Other compilers can simply be substituted in place of cc unless otherwise mentioned.

In brief the compilation process consists of the following steps :

Preprocessor - The Preprocessor accepts source code as input and is responsible for interpreting special preprocessor directives denoted by #. Like #include statements and #define statements.

C compiler - The compiler examines the source code for any syntax errors. Any errors discovered will be reported on the screen and the compilation ends. Logical errors are not detected by the compiler. Once the user has fixed the errors and re-compiled the program, the compiler translates the source code to assembly language code.

Assembler - Under UNIX, a seperate assembler processes this assembly code into a binary file. This is then written to a file with a .o extension on a UNIX system or as a .OBJ file on MS-DOS.

Linking - If a source file uses standard library functions or functions defined in other source files the linker combines these functions with the main() to create an executable file. External variable references are resolved here. This process is once again performed automatically when a cc command is issued under UNIX. The executable file in MS-DOS or Windows is the same as the source file name with a .exe extension.

Note : To change the default a.out on UNIX use the following compiler option with the cc command :

```
cc -o prg1 prg1.c
```

This creates an executable by name prg1.exe.

Executing a C program

In order to run or execute a C program just type a.out or if you have renamed the a.out by using the above compiler option then type prg1. This will effect the loading of the program into the computer's memory and initiating its execution. Any runtime errors like division by zero are now encountered. The program needs to be re-analysed for fixing the bugs, and the entire process of compiling, linking and executing should be repeated.

Lint

Before compiling a lengthy C program it is advisable to verify it by using a UNIX utility named *"Lint"*. This can be used as follows :

```
lint prg1.c
```

This utility does not create an executable file, but generates a list of possible bugs in the program. *Lint* performs type checking of variables and function assignments, checks for efficiency, unused variables and function identifiers, unreachable code and possibly memory leaks. It will also automatically check the program against Standard C Library to ensure that the arguments match in number and in type to any routine used from the library.

Make

Make is one of UNIX tools which simplifies compilation of complex programs or multiple programs. *Make* first looks for a file called "Makefile", if not found then a file called "makefile". Makefile has a number of components, the details of which are not covered in this chapter. Refer to the appendix for more details on this.

If your C program is a single file, you can usually use *make* by simply typing

```
make prog1
```

This will compile prog1.c, and put the executable code in prog1.

When a project consists of multiple files, you can compile each C file separately and link them into a program:

```
cc -c hello.c
cc -c goodbye.c
cc hello.o goodbye.o -o hello
```

Separate compilation is faster when you're debugging your program, because you only have to recompile the source code files that you modify. When you use *make* to manage your project, it keeps track of this for you and only recompiles the modules you need.

Source : http://www.peoi.org/Courses/Coursesen/cprog/frame2.html