Creating a Simple Computer Program Using the C Language

Creating a simple executable program using the C language is a straight-forward process. This process can be broken down into four processes:

1. Edit — Create or edit a source file using a text editor.
2. Preprocess — Replace macros, process header files, and remove comments.
3. Compile — Check the validity of the code and translate the source (text) file into machine (binary) code.
4. Link — Combine other object code and libraries to make one final executable program.
5. Execute — Run the program making sure it functions correctly.

These processes are commonly referred to as the edit/preprocess/compile/link/execute loop. ***Figure 1*** on the next page shows the flow chart of this process. It is called a loop because if the execution of the program (step five) does not work properly, you must start at step one and repeat the loop.

Step1: Editing
The first step is to create a file using a text editor. (Do not use a word processor, such as Microsoft Word. Word processors save files in their native format which is incompatible with programming tools.) The file you create should end with a **.c** extension and is referred to as your *source* file. Every Windows machine has a program called Notepad which can be used to edit source files. However, Notepad is a very rudimentary text file editor and is not recommended for daily use. The computers at DigiPen have a program called Notepad++, which is a much more powerful and capable text editor. This is the recommended editing tool for this class. However, any text editor (other than Notepad) can be used. Once you have typed in your code and saved it, you will now be ready for the next step: preprocessing.

Step2: Preprocess
The next step is to prepare the source file for the compiler. This is done with a program called a *preprocessor*. It performs a variety of tasks, most notably, it finds and includes header files, replaces macros in the code, and removes comments and extraneous white space. Essentially, any line that begins with a pound sign, #, is considered a *directive* and is acted upon by the preprocessor. The compiler never sees these directives or comments. Most of the time you won't manually perform this action. Instead, you will just jump to step 3 and the compiler will implicitly invoke the preprocessor itself before compiling the code. I'm including it here because it is a required step that is always performed.

Step 3: Compiling
The next step is to convert the (preprocessed) English-like text file into a language that the computer understands. The computer has a hard time with the English language so a program called a *compiler* is used to translate (or compile) the English-like C source file into machine code. This *machine* code, or *object* code, is in the computer's native language. A program called **gcc** (GNU C compiler) can be used to compile source code into object code. There is also a program called **g++**, which is the GNU C++ compiler, used for C++ source code. To compile your C source file, simply type:

```
gcc -c myfile.c
```

in the command window, where **myfile.c** is the name of your source file. That's all there is to it. You have now created a new file called **myfile.o**. If you were to look at this new file with your text editor, it would look like garbage on the screen. That's because it is now in machine code (binary), which doesn't make much sense to humans (or text editors). If you received any error messages while you were compiling, then you have some bad code that needs to be fixed. The compiler will tell you which lines in your text file caused the offending error. Simply load your source file back into your text editor, make the necessary changes, and compile your program again. Continue this process until the compiler stops complaining about your code. Now, you're ready for step four: linking.

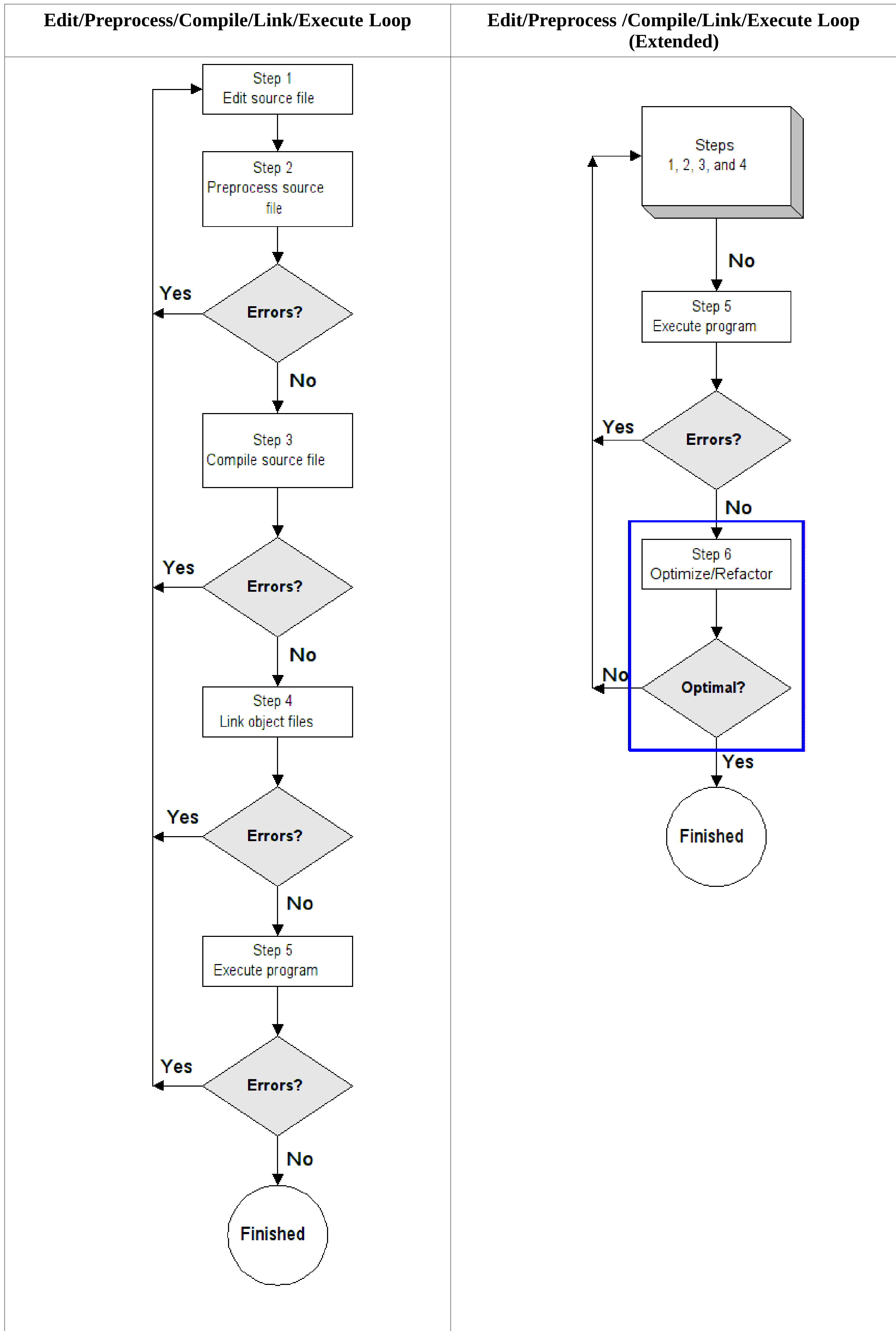

FIGURE 1 Flow Chart Representing the Edit/Preprocess/Compile/Link/Execute Loop

Step 4: Linking
Linking is the process by which your object code gets "connected" with other necessary code to create the final executable program. All programs use some kind of *external* code. This other necessary code could be in the form of standard libraries, other people's code, or some additional code of your own. Linking can be done automatically, (immediately after compiling) if your code is error-free. Unless you explicitly tell the compiler not to do any linking, most C compilers will automatically link your program after they have successfully compiled it. That's why you used a **`-c`** in step two above when you compiled your source code. This tells the compiler to compile only, with no linking. Now, to link your object file with other external code, simply type:

```
gcc myfile.o
```

Although I said that **`gcc`** was a compiler, it is, in fact, a front-end to the compiler *and* linker. Also, GCC no longer stands for GNU C Compiler; it stands for GNU Compiler Collection (because it can compile several different languages (such as C, C++, Java, Ada, etc.) **`gcc`** is smart enough to see that you are supplying an object file (the **`.o`** extension), so it knows that it should invoke the *linking* phase and not the *compiling* phase. If you want to compile and link with a single command, simply remove the **`-c`** from the command. This tells the compiler to link your program after it has been successfully compiled:

```
gcc myfile.c
```

Notice the missing **`-c`** switch and the **`.c`** extension. If any errors were encountered during the compile phase, the linker would not be invoked. If the linker found any errors during the linking process, you will have to make the necessary fixes to your source code and repeat the process. If everything is successful, you will now have another new file. This file will be called **`a.exe`**. This is an executable file, ready to run and is the default name given to programs that have been compiled and linked successfully.

Step 5: Executing
Executing is the easiest step. All you need to do is type **`a.exe`** and the program will execute. The hard part is testing whether or not the program is executing correctly, especially if the program is a non-trivial program. If you made a syntax mistake in your source file, the compiler will complain loudly and immediately. If you made a logic mistake in your thinking, you could spend many sleepless nights figuring out why things aren't working right. If you can prove that your program works correctly in all cases, your work is done. If not, you must start at step one again and repeat the process until the program runs correctly.

Summary
This process is basically quite simple. Most of the work is done by the computer, translating your source code into machine code. At this stage, knowing the intimate details about what the compiler and linker are doing are not necessary. As we move forward, we will encounter problems that can only be solved if you understand what is going on "behind the scenes." The important thing to understand for now is what steps are taken and the order in which these steps occur.