Creating C Programs

There are three fundamental stages, or processes, in the creation of any C program:

- Editing
- Compiling
- Executing

Editing

Editing is the process of creating and modifying C source code — the name given to the program instructions you write.

You can use any integrated development environment (IDE) such as Eclipse or Visual Studio Code for writing, managing, developing, and testing your programs.

You can also use a general-purpose text editor to create your source files, but the editor must store the code as plain text without any extra formatting data embedded in it.

If you're working with Linux, the most common text editor is the Vim editor. Alternately you might prefer to use the GNU Emacs editor. With Microsoft Windows, you could use one of the many freeware and shareware programming editors. These will often provide help in ensuring your code is correct, with syntax highlighting and autoindenting. There is also a version of Emacs for Microsoft Windows. The Vi and VIM editors from the UNIX environment are available for Windows too, and you could even use Notepad++.

Compiling

Compiling C code involves several stages, each responsible for a specific aspect of the translation process from human-readable source code to machine-executable code. Here are the typical stages of compiling C code:

1. Preprocessing:

   In this stage, the preprocessor processes the source code. The preprocessor directives, which begin with a # symbol, are executed. Common preprocessing tasks include including header files (#include), macro expansions, and conditional compilation (#ifdef, #ifndef, #endif, etc.). The output of this stage is often referred to as the "preprocessed code. You can see the preprocessor with -E compiler option.

2. Compilation:

   The preprocessed code is then passed to the compiler. The compiler translates the C code into assembly code or an intermediate representation. This stage involves lexical analysis, syntax analysis, semantic analysis, and optimization. The output is typically an object file (often with a .o or .obj extension), which contains machine code in a relocatable format.

   We will see compilation in greater details in future.

3. Assembly:

The object file generated in the compilation stage contains machine code in a format that is specific to the target architecture. The assembly stage translates this machine code into the actual machine code instructions for the target platform. The output is an assembly file or an executable file, depending on whether the code is intended to be part of a library or a standalone program.

4. Linking:

   In practice, a program of any significant size will consist of several source code files, from which the compiler generates object files that need to be linked. By breaking it up into a number of smaller source files that each provide a coherent part of what the complete program does, you can make the development of the program a lot easier. The source files can be compiled separately, which makes eliminating simple typographical errors a bit easier.

   The linking stage combines multiple object files (and possibly libraries) into a single executable file. It resolves symbols (variables and functions) and addresses, ensuring that references between different parts of the code are correctly linked. The output is an executable file with a binary format specific to the target platform (e.g., ELF on Linux, PE on Windows).

   The linking process involves two main types: static linking and dynamic linking. In static linking, all necessary code is combined into a single executable file. In dynamic linking, some code may be left in separate dynamic-link libraries (DLLs) or shared libraries (.so), and the linking occurs at runtime.

Executing

The execution stage is where you run your program, having completed all the previous processes successfully. Unfortunately, this stage can also generate a wide variety of error conditions that can include producing the wrong output, just sitting there and doing nothing, or perhaps crashing your computer for good measure. In all cases, it’s back to the editing process to check your source code.

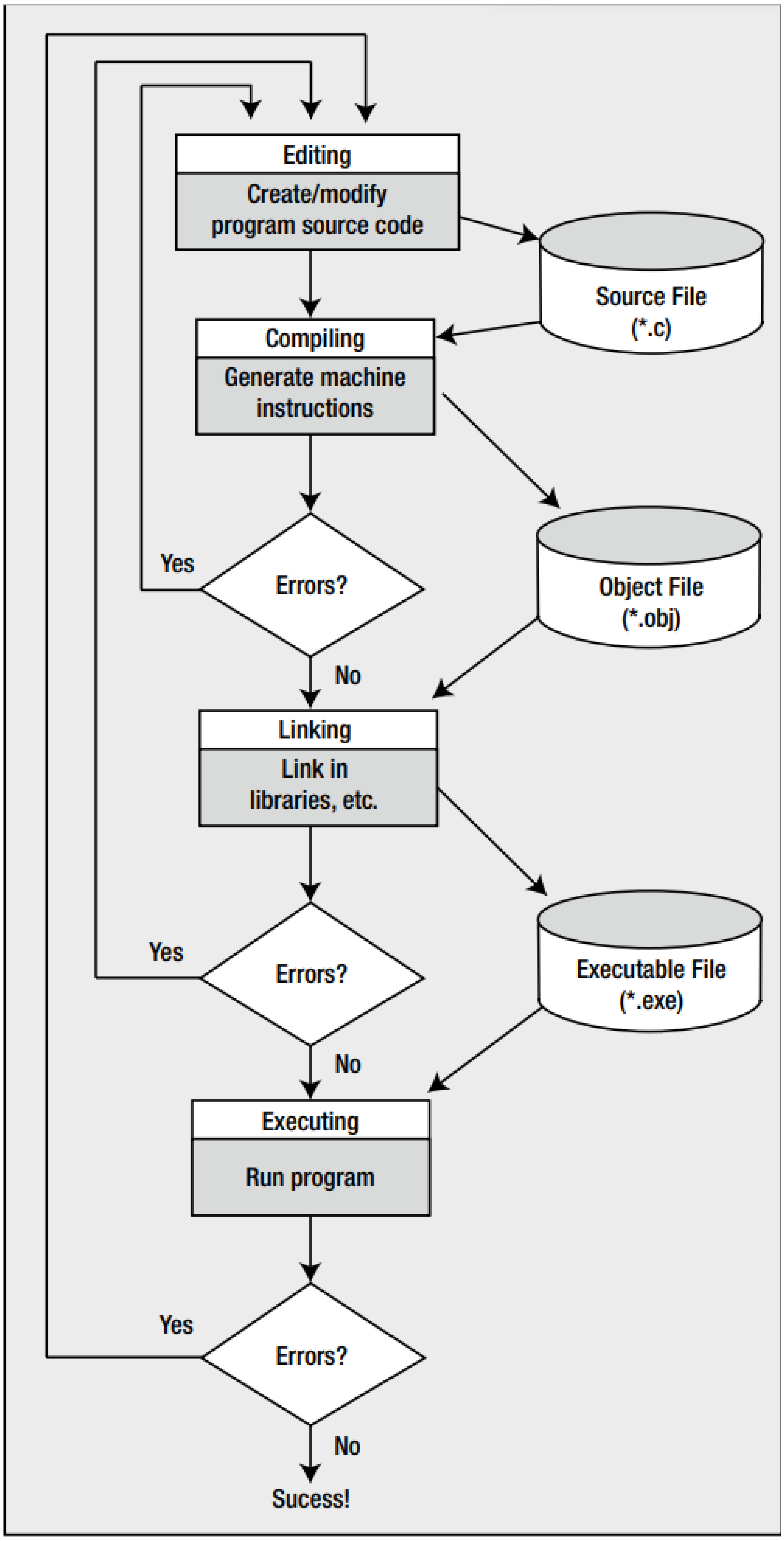
Now for the good news: this is also the stage where if your program works, you get to see your computer doing exactly what you told it to do!

In UNIX and Linux you can just enter the name of the file that has been compiled and linked to execute the program.

In most IDEs, you’ll find an appropriate menu command that allows you to run or execute your compiled program.

In Windows, you can run the .exe file for your program as you would any other executable.

The processes of editing, compiling, linking, and executing are essentially the same for developing programs in any environment and with any compiled language. Figure below summarizes how you would typically pass through processes as you create your own C programs.


Editing
Create/modify program source code
Source File (*.c)
Compiling
Generate machine instructions
Object File (*.obj)
Yes
Errors?
No
Linking
Link in libraries, etc.
Executable File (*.exe)
Yes
Errors?
No
Executing
Run program
Yes
Errors?
No
Sucess!

Creating Your First Program

Traditionally "Hello World!" is the first program in learning any language. We will also follow the same.

Edit

You can use any editor. Run your editor and type in the following program exactly as it's written.

```
/* Very First C Program - Displaying Hello World */
#include <stdio.h>
int main(void)
{
    printf("Hello world!");
    return 0;
}
```

When you've entered the source code, save the program as hello.c. This(.c) extension is the common convention when you write C programs and identifies the contents of the file as C source code.

compile

In order to compile the you need a compiler.

On Linux install the gcc compiler. On Windows we can use MingW which is gcc built to work with Windows. You can get it from this link https://sourceforge.net/projects/mingw/ and install the same.

Below command will generate an a.exe file which you can execute:

```
gcc .\hello.c
```

If you want to see the preprocessed file(hello.i) execute:

```
gcc -E .\hello.c -o hello.i
```

If you want to see the object file(hello.o) execute:

```
gcc -c .\hello.c
```

Execute

In order to execute the program run below command:

```
.\a.exe
```

It will print:

```
Hello world!
```