How do C Programs Run?

When starting software development in C, some of the most frequent questions you probably have are how to write and execute your programs. Common questions include: Where do I write my code? How is it converted into an executable? How do libraries work? Let's see about all these step by step to understand how C programs actually run.

Writing Your Code

You can write your C programs in two ways:

1. Using Text Editors

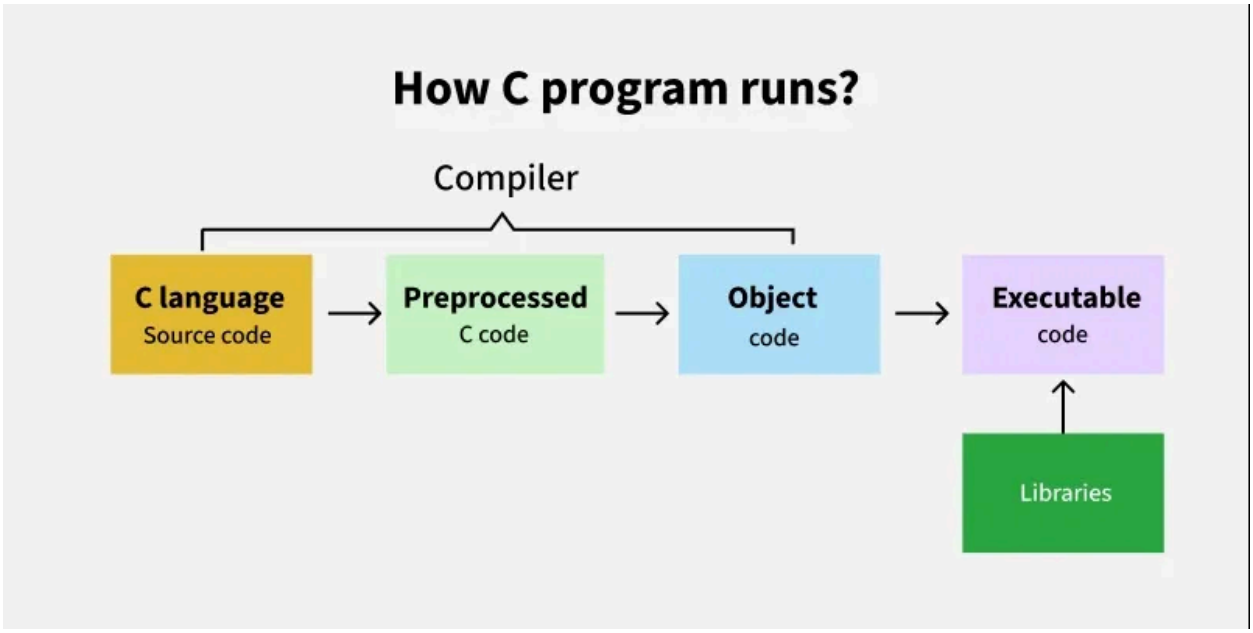
Text editors such as notepad in Windows and VI in Linux/Unix allows you to write the code but need manual compilation and executton of the code via the command line.

2. Integrated Development Environments (IDEs):

IDEs such as Code::Blocks, Visual Studio, Sublime Text, Dev-C++, and some online IDEs like GeeksforGeeks, all allow you to write and compile the code in one application.

They also use colors for syntax highlighting and provide quick buttons for compilation, execution, and debugging of your code. Because IDEs make the process faster and more convenient, that is why they are widely used in software development.

If you use a traditional text editor, you'll have to save your code and then invoke the compiler manually from the command line to compile and run it.



The Compilation Process

Once you've written your C code, it needs to be converted from human-readable code (high-level language) to machine-executable code. This process involves three key stages: Preprocessing, Compilation, and Linking.

1. Preprocessing

The preprocessor is the first step in the compilation process. It processes directives in your code that start with #, such as #include for including header files that contains declarations for library functions, global variables, and other components.

Other preprocessing tasks include processing macros, which you might have defined in your code. At this stage, your code is converted into pre-processed C code. It still looks like C code but now includes the declarations from the header files.

2. Compilation

The compiler then converts the pre-processed C code into object code, which is a binary representation of your program. It contains the machine instructions for your program. It still includes the declarations from the header files but not the actual implementation of library functions.

The compiler ensures your code is syntactically and semantically correct before converting it into object code.

3. Linking

After compilation, the **linker** comes into play. Its role is to combine your object code with the implementations of any external functions or libraries your program uses. The linker ensures all external dependencies (such as functions and variables from libraries or other modules) are resolved, creating a complete executable file.

For example, if your program uses printf to print output, the linker adds the actual implementation of printf from the standard library to your executable file.

Running Your Program

After the compilation and linking process, you get an executable file. You can run this file directly from the command line or within your IDE to see the output of your program.