How to Learn C

There is only one way to learn how to program and that is to write programs. You'll learn a lot more by writing and *debugging* programs than you ever will by reading about them. This means you will complete many programming exercises. When you do the exercises, keep good programming style in mind. Always comment your programs, even if you're only doing the exercises for yourself. Commenting helps you organize your thoughts and keeps you in practice when you go into the real world. Code that looks obvious to a programmer as he writes it is often confusing and cryptic when he revisits it a week later. Writing comments helps you to get organized before you write the actual code. If you can write out an idea in English, you're halfway to writing code in C.

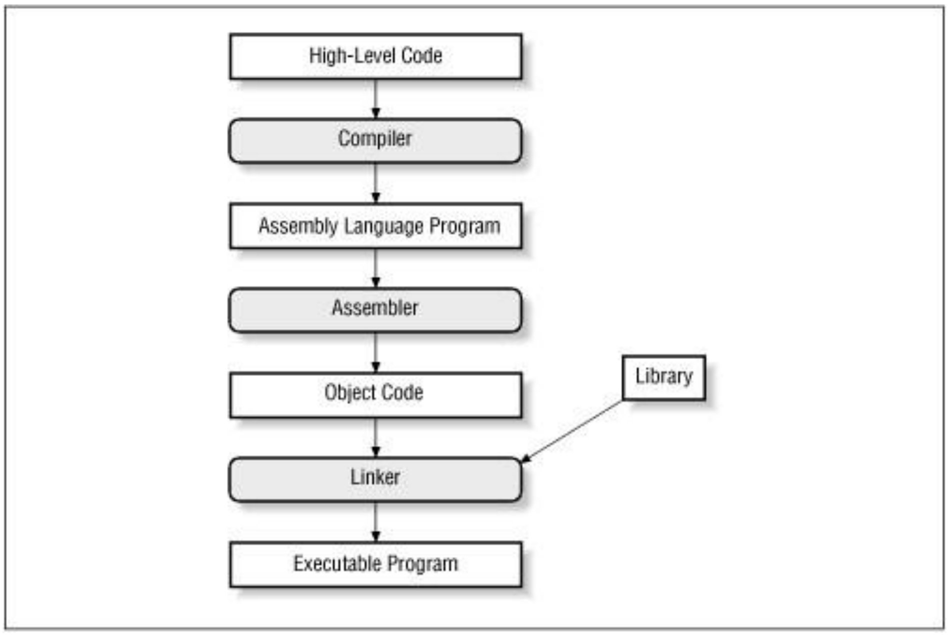
Throughout the course you will be exposed to many broken programs. Spend the time to figure out why they don't work. Often, the problem is very subtle, such as a misplaced semicolon or the use of = instead of ==. These broken programs will teach you how to spot mistakes in a small program. Then, when you make similar mistakes in a big program, and you *will* make mistakes, you will be trained to spot them.

Basics of Program Writing

Programs start as a set of instructions written by us. But before they can be used by the computer, they must undergo several transformations.

C programs are written in a high-level language using letters, numbers, and the other symbols found on a computer keyboard. Computers actually execute a very low-level language called *machine code* which is a series of numbers. So, before a program can be used, it must undergo several transformations.

Programs start out as an idea in a programmer's head. He/She then uses a text editor to write thoughts into a file called a *source file*, containing the *source code.* This file is then transformed by the *compiler* into an *object file*. Next, a program called the *linker* takes the object file, combines it with predefined routines from *standard libraries,* and then produces an *executable program* (a set of machine-language instructions the computer understands).



Transformation of a high-level language into a program.

Fortunately, you don't have to run the compiler, assembler, and linker individually since most C compilers use "wrapper" programs that determine which tools need to be run. Some programming systems go even further and provide the developer with an Integrated Development Environment (IDE) such as Visual Studio 2019. The IDE contains an editor, compiler, linker, project manager, debugger, and more all combined into one convenient package.

In the Assignments Folder you will find handout called “Visual Studio 2019 C Language Project Creation” that describes how to start creating our own programs. You will follow these instructions closely until you become familiar with this IDE.

Program Style
You must use good programming style to create simple, easy-to-read, programs. Discussing style before we know how to program might seem backward, but style is a very important part of programming.

Contrary to popular belief, programmers do not spend most of their time writing programs. Far more time is spent maintaining, upgrading, and debugging existing code than is ever spent on creating new programs. Most new software is written using existing source code.

Some programmers also believe that the purpose of a program is only to present the computer with a compact set of instructions. This concept is not true. Programs written only for the machine have two problems:

They are difficult to correct because sometimes even the author does not understand them.

Modifications and upgrades are difficult to make because the maintenance programmer must spend a considerable amount of time figuring out what the program does from its code.

Ideally, a program serves two purposes: first, it presents the computer with a set of instructions, and second, it provides the programmer with a clear, easy-to-read description of what the program does.

Comment Basics
Multi-Line comments in C start with a forward slash asterisk (/*) and end with an asterisk forward slash (*/) while single-line comments can start anywhere on the line with a double forward slash //.

The program comment sections should read like an essay. They should be as clear and easy to understand as possible. Good programming style comes from experience and practice.

The following comment sections give a brief description of suggested comment areas.

Heading: The first comment should contain the name of the program. Also include a short description of what the program does. You may have the most amazing program, one that slices, dices, and solves all the world's problems, but the program is useless if no one knows what it does.

Author: You've gone to a lot of trouble to create this program, take credit for it. Also, anyone who has to modify the program can come to you for information and help if needed.

Purpose: Why does this program do?

Usage: In this section, give a short explanation of how to run the program - every program would come with a set of documents describing how to use it.

References: Creative copying is a legitimate form of programming (if you don't break the copyright laws in the process). In this section, you should reference the original author of any work you copied.

File formats: List the files that your program reads from or writes to with a short description of their formats.

Restrictions: List any limits or restrictions that apply to the program, such as "The data file must be correctly formatted." or "The program does not check for input errors."

Revision history: This section contains a list indicating who modified the program, when it was modified, and what changes were made.

Error handling: If the program detects an error, describe what the program does with it.

Notes: Include special comments or any other information that has not already been covered that is important.

Note that the format of the comment sections will depend on what is needed for the environment in which you are programming.

Common Coding Practices

A *variable* is a place in the computer's memory used for storing a value. C identifies that place by the variable name. Names can be of any length and should be chosen so their meanings are clear. (Actually, a length limit exists, but it is so large that you probably will never encounter it.) Every variable in C must be declared. The following declaration tells C that three integer (int) variables are needed: p, q, and r.

```
int p,q,r;
```

But what are these variables used for? The reader has no idea.

Now consider another declaration:

```
int account_number;
int balance_owed;
```

Now we know the variables are used for an accounting program, but we could still use some additional information. For example, is the balance_owed in dollars or cents? We should add a comment after each declaration to explain what we were doing.

For example:

```
int account_number; /* Index for account table */
int balance_owed;   /* Total owed us (in pennies)*/
```

By putting a comment after each declaration, a mini-dictionary is created where we define the meaning for each variable name. Because the definition of each variable is in a known place, it's easy to know the meaning of a name.

You should take every opportunity to make sure that your program is clear and easy to understand.

Indentation and Code Format

In order to make programs easier to understand, most programmers indent their programs. The general rule for a C program is to indent one level for each new block or conditional statement.

Clarity

A program should read like a technical paper. It should be organized into sections or paragraphs. You should begin a paragraph with a topic-sentence comment and separate the comment from other paragraphs with a blank line.

Simplicity

Single functions should not be longer than a couple pages or it should be split into two simpler functions. Avoid complex logic like multiple nested control statements - the more complex your code, the more indentation levels you will need.

Make your program as simple and easy to understand as possible, even if you must break some of the rules. The goal is clarity, and the rules given in this section are designed to help you accomplish that goal.

Also, everything should be documented with comments. Comments serve two purposes. First, they tell the programmer how to follow the code, and second, they help the programmer remember what he did.