

It's the time to
write, such as a
program that
lets you learn
to avoid similar
mistakes and
spot them.

Qualline, Practical C Programming
3rd edition, O'Reilly 1997

In this chapter:

- *Programs from Conception to Execution*
- *Creating a Real Program*
- *Creating a Program Using a Command-Line Compiler*
- *Creating a Program Using an Integrated Development Environment*
- *Getting Help on UNIX*
- *Getting Help in an Integrated Development Environment*
- *IDE Cookbooks*
- *Programming Exercises*

2

Basics of Program Writing

The first and most important thing of all, at least for writers today, is to strip language clean, to lay it bare down to the bone.

—Ernest Hemingway

Programs start as a set of instructions written by a human being. Before they can be used by the computer, they must undergo several transformations. In this chapter, we'll learn how to enter a program, transform it into something the machine can use, and run it. Detailed steps are provided for the most popular UNIX and DOS/Windows compilers.

Programs from Conception to Execution

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

Programs start out as an idea in a programmer's head. He uses a text editor to write his thoughts into a file called a *source file*, containing *source code*. This file is transformed by the *compiler* into an *object file*. Next, a program called the *linker* takes the object file, combines it with predefined routines from a *standard library*, and produces an *executable program* (a set of machine-language instructions). In the following sections, we'll see how these various forms of the program work together to produce the final program.

Figure 2-1 shows the steps that must be taken to transform a program written in a high-level language into an executable program.

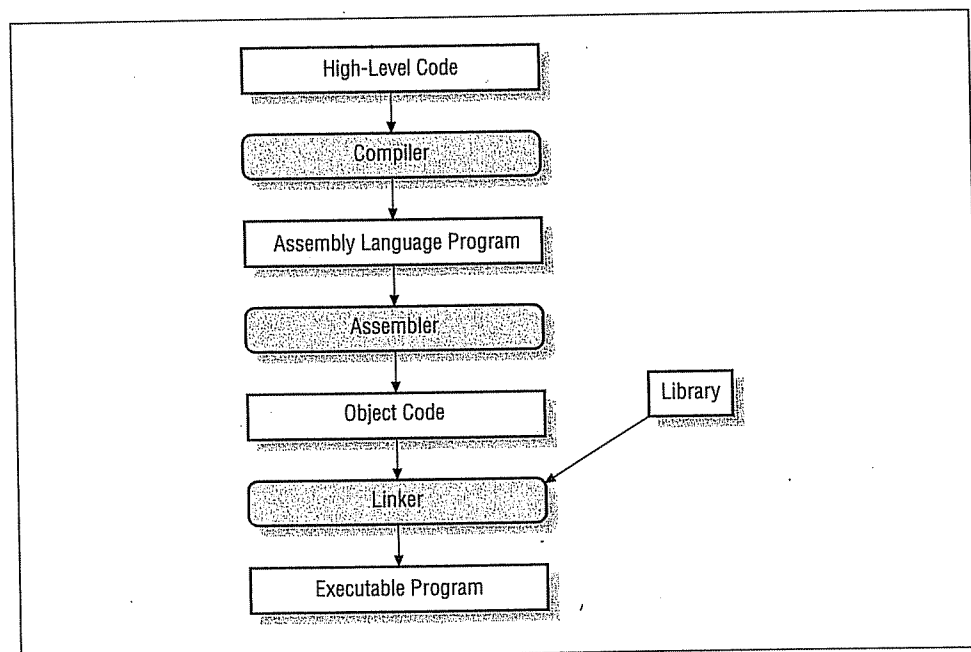


Figure 2-1: Transformation of a high-level language into a program

Wrappers

Fortunately you don't have to run the compiler, assembler, and linker individually. Most C compilers use "wrapper" programs that determine which tools need to be run and then run them.

Some programming systems go even further and provide the developer with an Integrated Development Environment (IDE). The IDE contains an editor, compiler, linker, project manager, debugger, and more in one convenient package. Both Borland and Microsoft provide IDEs with their compilers.

Creating a Real Program

Before we can actually start creating our own programs, we need to know how to use the basic programming tools. In this section, we will take you step by step through the process of entering, compiling, and running a simple program.

We will describe how to use two different types of compilers. The first type is the standalone or command-line compiler. This type of compiler is operated in a

Creating a Pr

batch mode
the compiler

The other ty
compiler, pr

Most UNIX :
available for
for MS-DOS.
compilers dc

Creativ Line Co

In this secti
program usi
UNIX comp:
Borland C++

However, if
skip ahead t

Step 1. Cr

You can mc
program tha
to hold our l

On UNIX typ

```
% mkdir
% cd hel
```

On MS-DOS

```
C:> MKDI
C:> CD E
```

Step 2. Cr

A program st

Example 2-1:

```
[File: hello
#include <st
```

* Turbo C++, Bc

batch mode from the command line. In other words, you type in a command, and the compiler turns your source code into an executable program.

The other type of compiler is contained in an IDE. The IDE contains an editor, compiler, project manager, and debugger in one package.

Most UNIX systems use command-line compilers. There are a few IDE compilers available for UNIX, but they are rare. On the other hand, almost every compiler for MS-DOS/Windows contains an IDE. For the command-line die-hards, these compilers do contain a command-line compiler as well.

Creating a Program Using a Command-Line Compiler

In this section, we'll go through the step-by-step process needed to create a program using a command-line compiler. Instructions are provided for a generic UNIX compiler (`cc`), the Free Software Foundation's `gcc` compiler, Turbo C++, Borland C++, and Microsoft Visual C++.*

However, if you are using a Borland or Microsoft compiler, you might want to skip ahead to the section on using the IDE.

Step 1. Create a Place for Your Program

You can more easily manage things if you create a separate directory for each program that you're working on. In this case, we'll create a directory called *hello* to hold our *hello* program.

On UNIX type:

```
% mkdir hello
% cd hello
```

On MS-DOS type:

```
C:> MKDIR HELLO
C:> CD HELLO
```

Step 2. Create the Program

A program starts out as a text file. Example 2-1 shows our program in source form.

Example 2-1: hello/hello.c

```
[File: hello/hello.c]
#include <stdio.h>
```

* Turbo C++, Borland C++, and Microsoft Visual C++ are all C++ compilers that can also compile C code.

Example 2-1: *hello/hello.c* (continued)

```
int main()
{
    printf("Hello World\n");
    return (0);
}
```

Use your favorite text editor to enter the program. Your file should be named *hello.c*.

WARNING MS-DOS/Windows users should *not* use a word processor such as MS-Word or WordPerfect to write their programs. Word processors add formatting codes to files, which confuse the compiler. You must use a text editor such as the MS-DOS "EDIT" program that is capable of editing ASCII files.

Step 3. Run the Compiler

The compiler takes the source file you've just made and converts it into an executable program. Each compiler has a different command line. The commands for the most popular compilers are listed below.

UNIX *cc* compiler (generic UNIX)

Most UNIX-based compilers follow the same generic standard. The C compiler is named *cc*, and to compile our hello program we need the following command:

```
% cc -g -ohello hello.c
```

The *-g* option enables debugging. (The compiler adds extra information to the program to make the program easier to debug.) The switch *-ohello* tells the compiler that the program is to be called *hello*, and the final *hello.c* is the name of the source file. See your compiler manual for details on all the possible options. There are several different C compilers for UNIX, so your command line may be slightly different.

Free Software Foundation's *gcc* compiler

The Free Software Foundation, the GNU people, publish a number of high-quality programs. (See the Glossary entry for information on how to get their software.) Among their offerings is a C compiler called *gcc*.

To compile a program using the *gcc* compiler use the following command line:

```
% gcc -g -Wall -ohello hello.c
```

Creating a Pr

The addition

The GNU *cc*
want to turn

```
% gcc -g
```

The switch -

The *-pedan*
feature it enc

Borland's *Tu*

Borland Inte
This compile
compile C c
C++ is:

```
C:> tcc
```

The *-ml tel*
number of d
the differenc
until you kno

The *-v switc*
ings are turn
Turbo C++ t
the source fil

You may
widely use
plex than

For exampl
the messa;

In Window
sages from
the messa;

You must
DOS or Ea

The additional switch `-Wall` turns on the warnings.

The GNU compiler contains several extensions to the basic C language. If you want to turn these features *off*, use the following command line:

```
% gcc -g -Wall -ansi -pedantic -ohello hello.c
```

The switch `-ansi` turns off features of GNU C that are incompatible with ANSI C. The `-pedantic` switch causes the compiler to issue a warning for any non-ANSI feature it encounters.

Borland's Turbo C++ under MS-DOS

Borland International makes a low-cost MS-DOS C++ compiler called Turbo C++. This compiler will compile both C and C++ code. We will describe only how to compile C code. Turbo C++ is ideal for learning. The command line for Turbo C++ is:

```
C:> tcc -ml -v -N -w -ehello hello.c
```

The `-ml` tells Turbo C++ to use the large-memory model. (The PC has a large number of different memory models. Only expert PC programmers need to know the difference between the various models. For now, just use the large model until you know more.)

The `-v` switch tells Turbo C++ to put debugging information in the program. Warnings are turned on by `-w`; stack checking is turned on by `-N`. Finally `-ehello` tells Turbo C++ to create a program named HELLO with `hello.c` being the name of the source file. See the Turbo C++ reference manual for a complete list of options.

Windows Programming

You may wonder why we describe MS-DOS programming when Windows is widely used. We do so because programming in Windows is much more complex than programming in MS-DOS.

For example, to print the message "Hello World" in MS-DOS, you merely print the message.

In Windows, you must create a window, create a function to handle the messages from that window, select a font, select a place to put the font, and output the message.

You must learn to walk before you can run. Therefore, we limit you to the MS-DOS or Easy-Win (Simplified Windows) programs in this book.

Borland C++ under MS-DOS and Windows

In addition to Turbo C++, Borland International also makes a full-featured, professional compiler for MS-DOS/Windows called Borland C++. Its command line is:

```
C:> bcc -ml -v -N -P -w -ehello hello.c
```

The command-line options are the same for both Turbo C++ and Borland C++.

Microsoft Visual C++

Microsoft Visual C++ is another C++/C compiler for MS-DOS/Windows. To compile, use the following command line:

```
C:> cl /AL /Zi /W1 hello.c
```

The /AL option tells the program to use the large memory model. Debugging is turned on with the /Zi option and warnings with the /W1 option.

Step 4. Execute the Program

To run the program (on UNIX or MS-DOS/Windows) type:

```
% hello
```

and the message:

```
Hello World
```

will appear on the screen.

Creating a Program Using an Integrated Development Environment

Integrated Development Environments (IDEs) provide a one-stop shop for programming. They take a compiler, editor, and debugger and wrap them into one neat package for the program.

Step 1. Create a Place for Your Program

You can more easily manage things if you create a separate directory for each program that you're working on. In this case, we'll create a directory called HELLO to hold our *hello* program.

On MS-DOS type:

```
C:> MKDIR HELLO
C:> CD HELLO
```

Creating a Program

Step 2. Enter the Project File Name

Each IDE is a little different, but the steps are the same.

Turbo C++

1. Start the Turbo C++ IDE.

```
C:> tc
```

2. Select the "File" menu option.

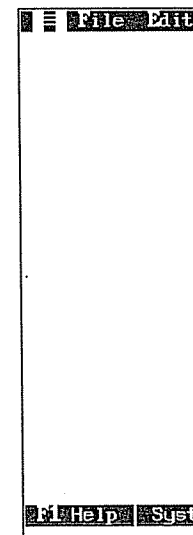


Figure 2-2: Clean

3. Select the "New" option.
 4. Select the "C++" option.
 5. Select the "Compiler M" option.
 6. Select the "warning me" option.
 7. Select the "project file" option.
- you're done