1 Displaying Text

1.1 Introduction

C is a modern, general purpose programming language created by Dennis Ritchie in the 1970's. It is used world-wide for a variety of applications. Examples of programs written in C include computer operating systems, company accounting systems and air-traffic control systems.

Many programs display something on the computer screen or monitor and our first program will do just this.

1.2 Displaying One Line of Text

The purpose of the first program in this chapter is to write *Good morning, sunshine.* on the screen. Here it is.

```
/* program 1.1 - displays a greeting on the screen. */
#include <stdio.h>

int main()
{
  printf("Good morning, sunshine.");
  return 0;
}
```

The line

```
/* program 1.1 - displays a greeting on the screen. */
```

is an example of a comment. A comment starts with /* and ends with */. A comment is written for the benefit of the person who has to read the program. The comment in this program tells us that the purpose of program 1.1 is to display a greeting on the screen. Another example of a comment is

```
/* Program written by: Terry Marris. */
```

The next line in the program is

```
#include <stdio.h>
```

stdio is the name of a standard C library. This library provides a utility for writing text on the screen. An example of such a utility is the function named *printf*. We want to use this function so we specify that the contents of the library is to be included in our program. *stdio* stands for standard input and output. An example of output is text displayed on a screen. An

example of input is data entered via a keyboard. We shall deal with input later. The *h* in *stdio.h* stands for header because the library is always included near the head of a program. There are other libraries; we shall deal with them in due course. The # symbol is an essential part of the include command as are the angle brackets surrounding *stdio.h*.

The next line is

```
int main()
```

main is a compulsory part of every C program. *main* is the point at which program execution starts. By program execution we mean that the instructions are carried out one after the other. In our program there is just one instruction; this is

```
printf("Good morning, sunshine.");
```

int stands for integer, a whole number such as 1, 2 or 3. We shall dicover its purpose when we discuss functions in chapter four.

Now we come to the lines

```
{
  printf("Good morning, sunshine.");
  return 0;
}
```

Here, the braces *{* and *}* mark the beginning and end of a block. A block contains a sequence of instructions. Instructions are also known as statements. In our example there are two statements: they are

```
printf("Good morning, sunshine.");
return 0;
```

printf stands for print formatter. We use *printf* to arrange and print text on the screen. The text to be printed here is contained between the quotation marks. The quotation marks themselves are not printed on the screen when the statement is executed, that is, when the *printf* instruction is carried out.

return 0 is a C idiom that informs the operating system (e.g. Windows or Linux) that the *main* function reports no problems. We discuss the return statement in Chapter four - Functions.

There are some important points to note about C programs. C words such as *include*, *int*, *main* and *return* must be written entirely in small letters, that is, in lower case. Library function names such as *printf* must be written entirely in lower case with no spaces between the letters. Every statement must end with a semi-colon. Notice that a semi-colon follows the *printf* statement in program 1.1. And we follow the convention that a block of statements is indented by two spaces in from the left hand margin. So, for example, using * to indicate one push of the spacebar, we write

```
{
**printf("Good morning, sunshine.");
**return 0;
}
```

1.3 Creating a Working Program

There are several steps to creating a working program. First, the program text has to be typed into the computer. An example of program text is program 1.1 shown above. A C program is entered with the aid of another program called an editor. An editor is usually provided as part of your computer's operating system. An editor allows you to create and correct program text.

Then the program text has to be translated into a code which enables the computer to perform the required tasks. The translation process is carried out by another program known as a C compiler. If the compiler finds, for example, an incorrectly written word or that a bracket is missing, it will fail to complete the translation process. If the compiler fails to complete the translation then a message indicating the error (or errors) is displayed on the screen. You should read these error messages before returning to the editor to correct the mistakes.

Then, when the translation has been successfully completed, the code has to be combined with code from libraries. These libraries contain functions which are responsible for managing devices such as a computer's screen. This process is achieved by yet another program known as a linker.

Finally, the completed program has to be loaded into a computer's memory and then executed.

Figure 1.1 summarises the steps required to create a working program.

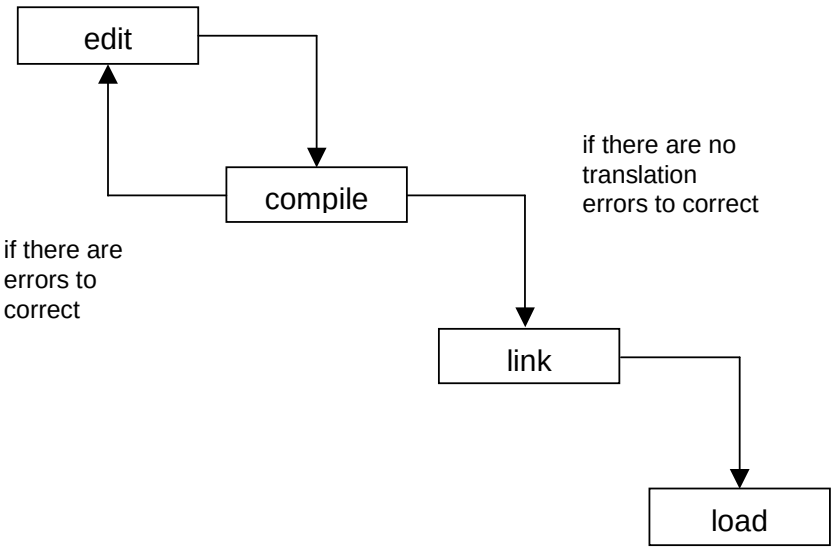
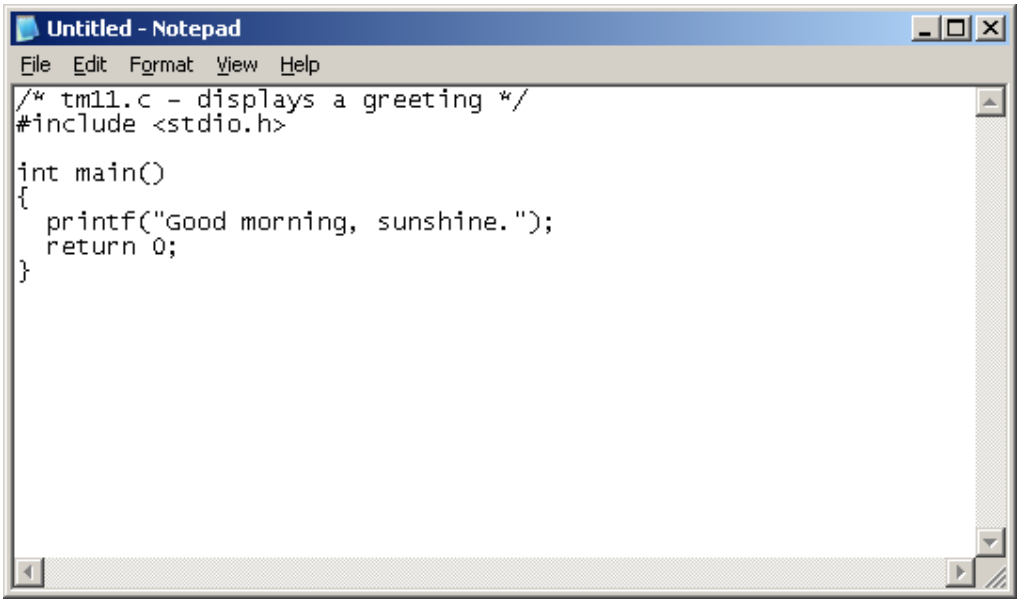


Figure 1.1 - the steps required to complete and execute (ie run) a C program.

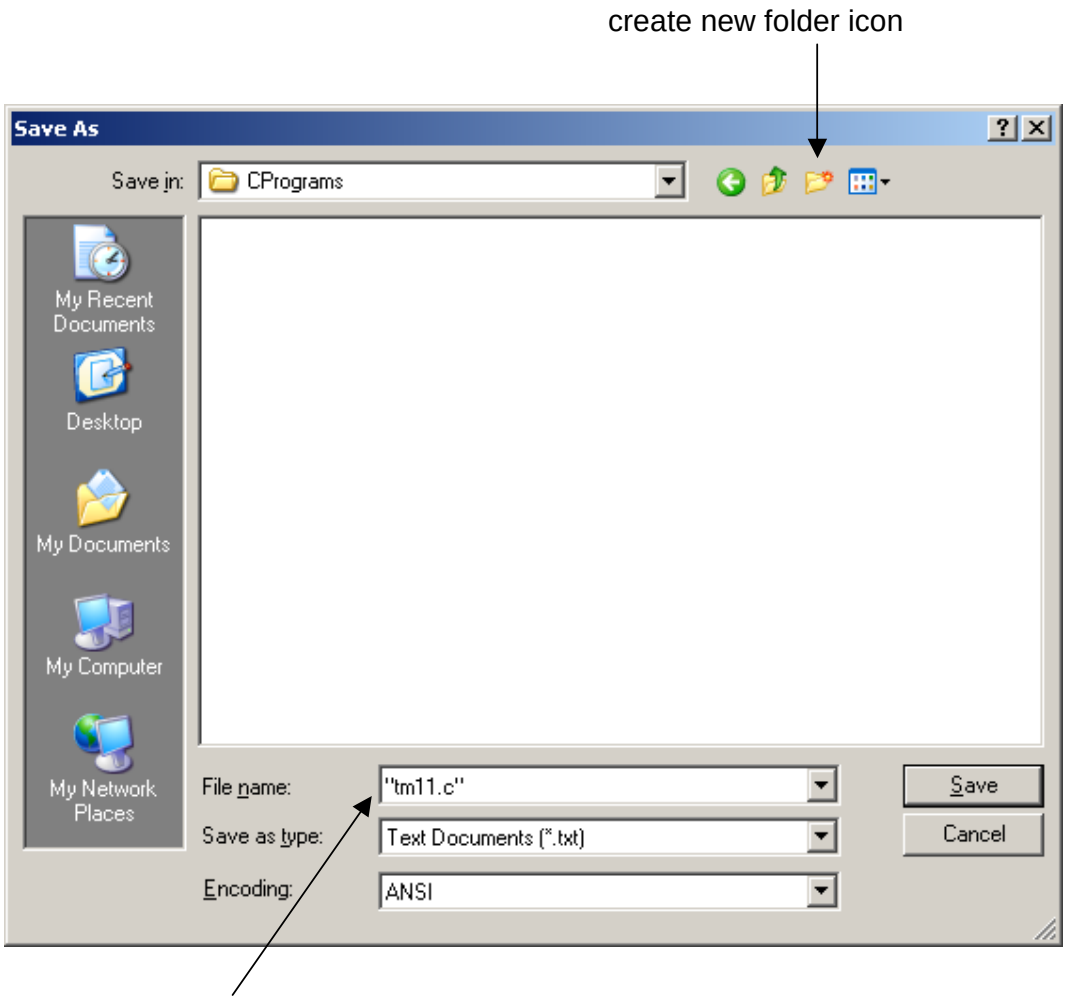
If you are using Microsoft Windows, one way of proceeding is:

1. Load Notepad: in Windows choose *Start*, *Programs*, *Accessories*, *Notepad*

2. Type in or edit your program text



3. Save your program in a folder of your choice: choose *File*, *Save As*, navigate to your folder (use the create new folder icon to create one if necessary). Enclose your file name in quotation marks. Avoid using spaces in your file names.

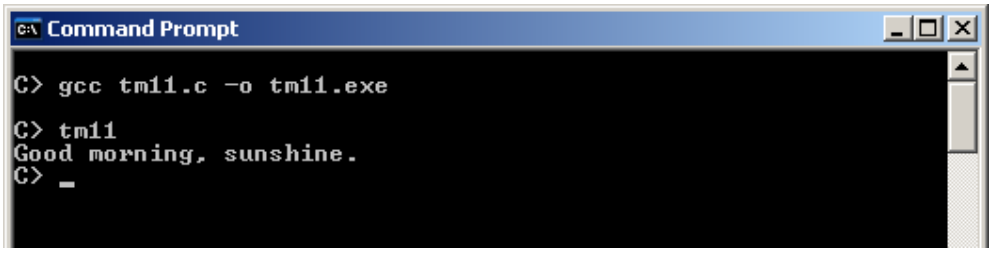


4. Load the Windows command prompt: in Windows choose *Start*, *Programs*, *Accessories*, *Comand Prompt*

5. Navigate to the directory or folder containing your C program e.g. *cd my documents\CPrograms*

6. Compile and link your program: *gcc tm11.c -o tm11.exe*

7. Run your program: *tm11*



The specific commands to edit, compile and link a program differ from one C programming system to another. So, you might need to consult the instructions that came with your C compiler or a local expert.

Exercise 1.1

1 Write and execute (that is, run) a program which will display, on the screen, the first line of a song or a poem. Use program 1.1 as a guide. You will need to pay careful attention to punctuation, to the way you use spaces and to the way you write C words such as void, main and include. C words are never written with capital letters.

2 A compile-time error occurs if the compiler fails to translate program text into code. When a compile-time error occurs, a message known as a diagnostic is output on the screen. Diagnostics inform the programmer about the nature of the problem. Write down four compile-time error messages output by the compiler, the program line which caused each error and how each error could be corrected. Hint: you could introduce some deliberate errors in the program you wrote for question one above.

1.4 Displaying Several Lines of Text

The purpose of the second program in this chapter is to display on the screen

```
Good morning, sunshine.
The world says "Hello!".
```

We want to display *The world says "Hello!".* on the next line below *Good morning, sunshine.*

To instruct our program to place text on a new line we write

```
printf("\n");
```

The effect of this statement is to move the screen cursor to the beginning of the next line down. (A screen cursor is usually a flashing line or block which tells you where you are on the screen.) *n* stands for the new-line character. Notice that \ is not the same thing as /. Shown below is a C program that prints two lines of text.

```
/* program 1.2 - displays two lines of text on the screen. */
#include <stdio.h>

int main()
{
  printf("Good morning, sunshine.");
  printf("\n");
  printf("The world says \"Hello!\".");
}
```

The line

```
printf("The world says \"Hello!\".");
```

prints *The world says "Hello!".* on the screen. If we want *printf* to print a quotation mark on the screen, then we write a backslash followed by a quotation mark like this: *\"*.

Look at the block of statements bracketed between the *{* and the *};* these statements are

```
printf("Good morning, sunshine.");
printf("\n");
printf("The world says \"Hello!\".");
```

We read the statement sequence like this:

do *printf("Good morning, sunshine.");*
and then do *printf("\n");*
and then do *printf("The world says \"Hello!\".");*

Notice that in program 1.2 the *printf* statements are written in line one under the other and that they are indented by two spaces in from the left hand margin. We use indentation to help us to read and understand large and complex programs.

Exercise 1.2

1. Write and run a program which will display, on the screen, four lines of a song or a poem.

1.5 Documentation

The purpose of documentation is to ensure a standard way of specifying and writing and describing programs. A standard way of working helps programmers to understand each others work. We shall adopt the following conventions.

Every program is to begin with comments which identifies the program, describes its purpose, names its author and states the date it was written or changed.

A space is to follow the character pair /* and precede */ to make the comments easy to read.

For example:

```
/* program 1.1 - displays a greeting. */
```

Always check that you end each comment correctly with */ and remember that you cannot have one comment inside another.

A block of statements sandwiched between a *{* and an *}* is to be indented by two spaces to help make the program structure clear to the (human) reader.

Every statement is to be written on a new line; the only exception to this is if several statements written on the same line makes the program easier to understand. Many statements written on the same line makes programs unnecessarily difficult to read and follow.

1.6 Programming Principles

Pay strict attention to layout at the time the program is being entered or edited; this helps to minimise the chances of making errors and makes it easier for others (such as teachers!) to see your mistakes.

Exercise 1.3

1 Usually, the process of producing an executable program results in several files being saved on your disk directory. Examine your disk directory and write down the names of the files which were produced as a result of creating and running your first program. Explain the origin of each file. Use Figure 1.2 to help you.

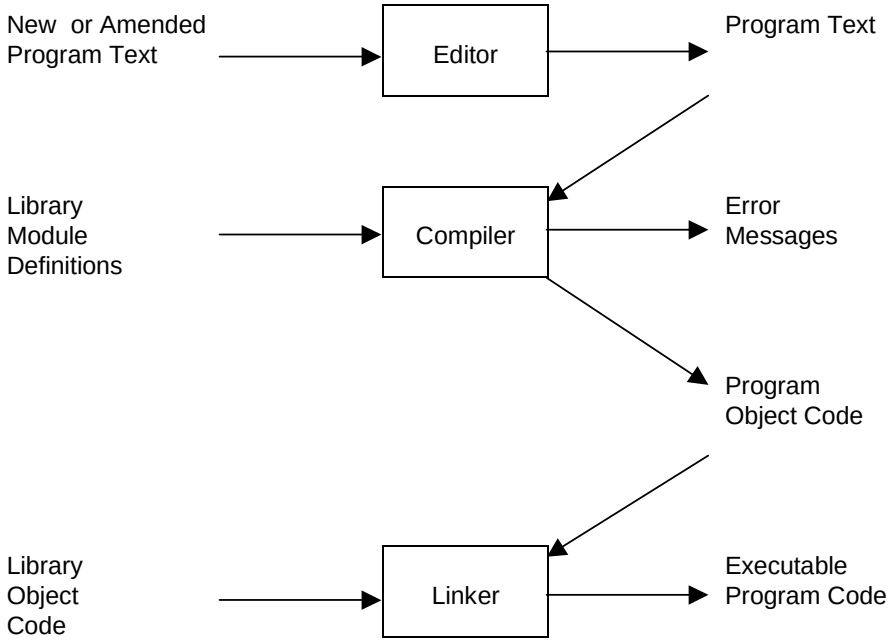


Figure 1.2 - the edit-compile-link process to produce an executable program. The

input to the compiler is program text (eg prog1.c) and library module definitions (eg stdio.h). The output from a compiler includes errors (if any) and object code (eg prog1.o). Object code is linked with library modules (eg c.lib) to form a complete and executable program (eg prog1.exe).