2 Chapter

Overview of C Language

OBJECTIVES

After reading this chapter, the reader will be able to

- appreciate the importance of C as a programming language
- understand the structure of a C program
- learn the preferred practices in writing a program
- understand the concepts of compilation, linking and loading of a C program
- learn the desirable characteristics of a good program

1. INTRODUCTION

The C programming language was developed by Dennis Ritchie at the Bell Laboratories in 1972. C evolved from two earlier languages called the BPCL and B which were also developed at Bell Laboratories. In 1978, Brian Kernighan and Dennis Ritchie published a definitive description of the language, referred to as the 'K&R C'. Because of its features, C became popular very fast and by 1980s it was one of the most popular programming language being used.

C is a general-purpose structured programming language. It has a rich set of data types and has a syntax that uses the English language keywords. Its features categorize it as a high-level language. C has additional features that allow it to be used at the lower level, thus bridging the gap between the machine language and the conventional high-level languages. This flexibility allows C to be widely used for systems programming.

C compilers are commonly available for computers of all sizes. The compilers are usually compact, and they generate object codes that are small and highly efficient as compared to programs compiled in other high-level languages.

An important characteristic of C is that the programs are highly portable. This is because C implements most computer dependent functions as its library functions. Every version of C is accompanied by its own set of library functions. The library functions are relatively standardized. Therefore, most C programs can be processed on many different computers with little or no alteration.

This book covers the features of C that are included in the ANSI standard and are supported by most commercial C compilers.

arguments and return type of a function will be discussed later in the chapter on User-defined Functions.

- The opening brace `{` in the second line of the program marks the beginning of the function `main()`. The closing brace `}` after the set of statements marks the end of the function `main()` and also the end of the program.
- The set of statements to be executed are written within the braces `{}` and is called the function body. It is responsible for performing a specific task. The statements may be in the form of declarations, expressions, assignment statements, predefined functions, control statements or compound statements. All these will be discussed in detail separately as we proceed our journey through C programming.
- The statement `printf("My first C program");` is an executable statement and on execution it displays `My first C program` on the screen. `printf()` is a predefined library function in C.
- The semicolon (`;`) at the end of the `printf()` statement is called a statement terminator. Every statement is a C program ends with a semicolon. A missing `;` generates a syntax error.
- The `#include<stdio.h>` is a preprocessor directive that includes the header file `stdio.h` in the program. The header file contains the pre-defined function `printf()` and is referred to use this function in the program. Any number of header files can be included in a program file depending upon the library functions used in the program. Header files are generally included at the beginning before the `main()`. We will study more about header files and preprocessor directives in the chapter on the C preprocessor.

Now we will modify the program in Illustration 1 to illustrate the use of comment or remark statements.

Illustration 2: Write a program to illustrate the use of comment statement.

```
#include<stdio.h>
void main()
{
/* Start of Printing*/
printf("My first C program");
/*End of Printing*/
}
```

- The line beginning with `/*` and ending with `*/` is called a comment line or a remark line. Comment lines are non-executable, and anything between `/*` and `*/` is ignored by the compiler. Comment lines are used in C programs to increase the understandability of the program. The comment lines can be inserted at the beginning of the program before the `main()` or within a function.

Illustration 3: Write a program to illustrate the use of comment lines at different places in a C program.

```
/*C program illustrating use of comment lines*/
#include<stdio.h>
```

```
void main()
{
/*Printing starts*/
printf("This is a book.");           /* executable statement*/
printf("I will read it");            /* executable statement */
/* Printing ends*/
}
```

Test Run:

```
This is a book. I will read it
```

Since comment lines are non-executable statements, they do not affect the execution speed of the programs and the size of the compiled program. Hence, we should use the comment lines liberally in programs as they help the programmers and others in understanding the program and serve as an aid in debugging the program and its maintenance.

Illustration 4: Write a program to print text in two lines.

```
#include<stdio.h>
void main()
{
printf("This is a book \n");
printf("I will read it");
}
```

Test Run:

```
This is a book
I will read it
```

Note the difference between the outputs of Illustrations 3 and 4.

The `\n` character (called the 'new line character') in Illustration 4 enables the printing in 2 lines i.e. the new line character `\n` causes the printing to shift to the next line. A new line is said to have been inserted.

Illustration 5: Write a program to print text in several lines.

```
#include<stdio.h>
void main()
{
printf("ABC\nCD\nEF");
printf("\n123\n456");
}
```

Test Run:

```
ABC
CD
EF
123
456
```

5. MORE SAMPLE PROGRAMS

We will discuss some more illustrations of C programs to understand C programming better at this level, though we will study the concepts used in these programs in detail in the later chapters.

Illustration 6: Write a program to compute and print the area of a circle given its radius.

```
/*Program to find area of circle*/
#include<stdio.h>
void main()
{
int radius=10;                  /*initialization  statement*/
float area;                     /*declaration statement*/
area=3.142*radius*radius;       /*assignment statement*/
printf("Area=%f", area);        /*output statement*/
}
```

Test Run:

```
Area=314.2000000
```

Illustration 7: Write a program to compute and print the area of a circle, its radius to be input by the user.

```
#include<stdio.h>
void main()
{
int r;
float area;
printf("Enter the radius of the circle");
scanf("%d", &r);
area=3.142*r*r;
printf("\nArea=%f", area);
}
```

Test Run:

```
Enter the radius of the circle 10
Area=314.2000000
```

Illustration 8: Write a program to compute the sum of two numbers input by the user.

```
#include<stdio.h>
void main()
{
int a,b,sum;
printf("Enter the values of a and b");
scanf("%d  %d", &a, &b);
sum=a+b;
printf("Sum=%d", sum);
}
```

(3) Linking the program to the functions from the header files
(4) Executing the program

Figure 2.2. shows the steps involved in writing, compiling, linking and executing a C program. The C program is created or written using the built-in editor of the C compiler. The Windows Notepad application can also be used to write C programs. The C program so written is called the source code and is saved with an extension `.c` or `.C` following the filename. For example `first.c`

The source code is given to the compiler for compilation. During the compilation process, the C compiler scans the source code for syntax errors and lists the errors detected.

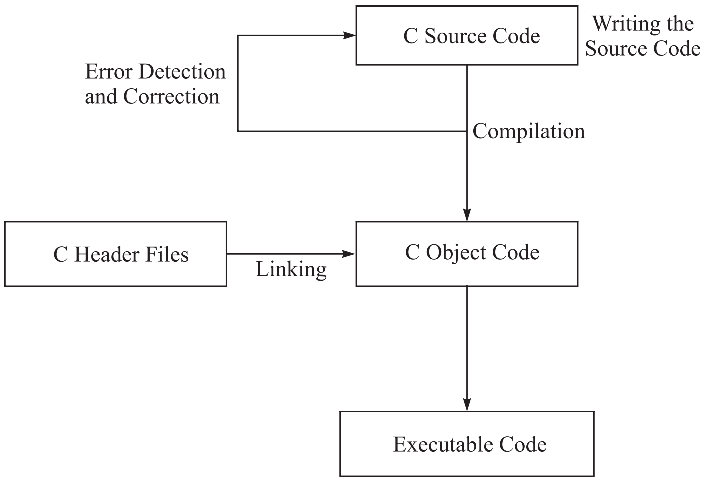

Fig 2.2 *The C Compilation Process*

The errors are required to be corrected by the user, and the corrected source code is again fed to the compiler. The compiler checks for errors and if errors still exist, they are listed by the compiler. The process of correcting the errors continues till the source code is error free. The compiler converts the source code into a binary form called the object code. The object code files have extension `.obj`.

The header files contained in the C library consist of the predefined functions for various standard tasks such as output, input, math functions, string manipulation functions and others. The functions with similar functionality are grouped into various header files. The header files are included at the beginning of the source code file using the preprocessor directive `#include`. The header files carry an extension `.h`.

The job of the linker is to link the header files with the object code to generate the binary executable files that have an extension `.exe` in the Windows operating system.

Sometimes a program may involve multiple source code files. In such cases, the individual source code files are compiled into object codes and all the object codes as well as the header files are linked by the linker to generate the executable file. This is shown in Fig. 2.3.

The software that brings the executable file residing on the hard drive to the main memory for execution is called the loader.

and clarity of the program. Sometimes complex programs may require a tradeoff between the efficiency and clarity. Experienced programmers can take proper decisions in such cases.

Modularity: Many programs involving multiple tasks can be broken down into subroutines or subtasks. Each subroutine is capable of performing a task and the various subroutines together constitute a complete program. This is called modularization and helps in enhancing the clarity and understandability of the program as well as providing means for future extension of the program by adding new modules. Hence, if possible, the program should be divided into appropriate modules or subroutines.

Generality refers to the ability to use the same program for a wide variety of data. For example, it may be more advisable to read/input the values of various data items rather than to fix their values in a program.

POINTS TO REMEMBER

- C is a middle-level language i.e. it has the capabilities of the low-level languages as well as the high-level languages.
- C is a highly portable language. A C program written for one computer or operating system can be run on another computer or operating system with little or no modification.
- C is a highly structured language and C programs are written as collection of modules or functions.
- C has a well-defined syntax or set of rules.
- Execution of a C program starts with the `main()` function.
- C is a case-sensitive language.
- Every C statement is terminated by a semicolon (`;`).
- Comment statements are non-executable statements.

REVIEW QUESTIONS

1. Discuss the importance of C as a programming language.
2. Why is C called a case sensitive language?
3. What is the purpose of using comment statements in a C program?
4. Discuss the structure of a C program.
5. Explain the process from writing a C program to its execution.
6. List the features of a preferred programming style.
7. What are the desirable characteristics of a good program?