13
A COMPLETE C PROGRAM

13.1 INTRODUCTION

It is necessary to do some planning before starting actual programming work. This planning will include the logic of the actual program. After logical approach of the program, the syntactic details of the language can be considered. This approach is often referred to as **"top-down"** programming. Top-down program organization is normally carried out by developing an informal outline, consisting of phrases or sentences that are part of English and part of C language. In the initial stages of program development, the amount of C is minimal, consisting only of various elements that define major program components, such as function headings, function references etc. Additional details is then provided by descriptive english material inserted between these elements, often in the form of program comments. The resulting outline is usually referred to as **pseudocode.** Another method sometimes used is the **"bottom-up"** approach. This approach involves the detailed development of these program modules early in the overall planning process.

13.2 OBJECTIVES

After going through this lesson you will be able to

- write a C program
- compile and execute the program
- detect and correct errors

13.3 WRITE C PROGRAM

Once the planning of a program has been formulated, the detailed working development of C program can be considered. This includes translation of each step of the program outline into one or more equivalent C instructions.

There is more to do to write a complete C program than simply arranging the individual declarations and statements in the right order. There are some additional features to be added to enhance the program's readability and its resulting output. There must be logical sequencing of the statements, the use of indentation, the use of comments and the generation of clearly labeled output statements and use of comments for enhancement of program readability.

The use of indentation is closely related to the sequencing of groups of statements within a group. It describes the subordinate nature of individual statements within a group. Comments should always be included within a C program. They can identify certain key items within the program and provide other useful information about the program. The other important characteristic of a well-written program is its ability to generate clear, legible output. When executing an interactive program, the user may not know how to enter the required input data. For example, the user may not know what data items are required, when the data items should be entered, or the order in which they should be entered. Thus a well written interactive program should generate prompts at appropriate times during the program execution in order to provide this information.

Let us consider an example of interactive program:

```
# include < stdio.h>
/ * simple calculation of average marks* /
main ( )
{
 float m1, m2, avg ;
/ * read input data * /
print ("please enter a value for marks m1 :\n");
scanf ("%f", & m1);
printf ("please enter a value for marks m2;");
scanf("%f", & m2);
avg = m1+m2 / 2 ;
/ * write output * /
printf ( "/n the final value (avg) is: %f ", avg);
}
```

The first line of this program is # include <stdio.h>, <stdio.h> is the file which has definitions of standard input output files, this file must be included in every C program as every program obviously needs printf and scanf statements. The second line, main ( ) function which is followed by curly braces. Now comes the definition and declaration of variables and functions as needed by the programmer. After that there is body of the program which is actually the syntactical conversion of logic of the program. The curly braces of the main function should be end with closed curly braces.

Once the program has been written it must be entered into the computer before it can be compiled and executed. This is usually accomplished by one of two possible methods, the most common being the use of an editor. Most computer systems include an editor that is used to create and alter text file. Some editors are line-oriented, while others are character-oriented. The commands within a line-oriented editor permit various editing operations, such as insert, delete, copy to be carried out on specified lines within the text files. Character-oriented editors are generally used with interactive terminals. They are also referred to as full-screen editors. Any editor can be used to enter a C program and its accompanying data files, though some C compilers are associated with operating systems that include their own editors.

Regardless of particular editor being used, the procedure for creating a new program is to enter the program into a text file line-by-line. The text file is then assigned a file name and stored either in the computer's memory or on an auxiliary storage device. Usually a suffix, such as c is attached to the file name, thus identifying the file as a C program. Such suffixes are called **extensions**. The program will remain in this form until it is ready to be processed by C compiler.

13.4 COMPILING AND EXECUTING THE PROGRAM

Once a complete program has been correctly entered into the computer, it can be compiled and executed. The compilation is usually accomplished automatically in response to a single command say *compile sample*, where sample refers to the name of the file containing the program (sample.c). It may also be necessary to link the compiled object program (sample.obj)with one or more library routines in order to execute the program. This is done by the command *link sample*. In the UNIX operating system the compile and link steps can both be carried out using the single instruction *cc sample*.c, resulting in an object program called sample.out. The successful compilation and linking of a C program will result in an executable object program that can then be executed in response to an appropriate system command, such *as execute*. In UNIX, the execution can be initiated simply by typing the name of the object program i.e. sample.out

Let us consider the previous example of program average of 2 marks, suppose the name of program is average. After writing the program correctly, compilation must take place. This is done by compiling average. The computer will respond by compiling the program, resulting in a non executable object program called average.obj. Typing errors, syntactic errors, and so on can easily be corrected by reentering the editor and making the required changes. If the compilation is carried out successfully, the next step is to link the program with the program library routines. This is accomplished with the command:-

```
link average
```

The result of the linking process will be the creation of an executable objet program, called average.exe. If the compilation has not been successful, it would not have been possible to proceed with the link step. The programmer must then find the source of error, re-enter the editor, and correct the original source program so that it can again be compiled. To exceute the final object program, simply type the program name e.g., average.out.

After this the following interactive dialog will be generated:

```
Please enter a value for marks m1: 50

Please enter a value for marks m2: 50

The final value (avg) is : 50
```

C interpreters translate a source program into object code on a line by line basis and then execute the newly generated object code directly. Thus the generation and execution of object code occurs simultaneously and there is no link step. Successful compilation, linking and execution of a C program often requires several attempts because of the presence of errors that are generally present in a new program.

Programming errors often remain undetected until an attempt is made to compile or execute the program. Some particular common errors of this type are improperly declared variables, a reference to an undeclared variable or incorrect punctuation. Such errors are referred to as syntactic or grammatical errors. Most C compilers will generate diagnostic messages when syntactic errors have been detected during the compilation process. Let us consider an example, which contains several syntactic errors.

```
include < stdio.h>
main
{
 float m1, m2, avg ;
/ * read input data * /
printf (“please enter a value for marks m1 :\n”);
scanf (“%f”, m1);
printf (“please enter a value for marks m2;);
scanf(“%f”, & m2);
avg = m1+m2 / 2 ;
/ * write output * /
printf ( (“\n the final value (avg) is: %f ”, avg);
```

After compiling, the errors are as follows:

1. The include statements does not begin with a # sign, main does not include a pair of parentheses.
2. The first scanf statement does not have an ampersand (&) preceding the argument.
3. The 2nd printf statement does not have a closing quotation mark.
4. The program does not end with a closing brace }

These errors are of general type, in the form of C compiler the messages are as follows:

```
Average.C(2): errors 54: expected ‘( ‘ to follow ‘include’
Average.C(7): errors 61: syntax error: identifier ‘scanf’
Average.C(8): errors 61: syntax error: identifier ‘printf’
Average.C(13): fatal error: unexpected EOF
```

The error message consists of the file name, followed by the line number where an error has been detected. This information is followed by a numbered reference to an error type, which is then followed by a very brief message. Another common type of error is the **execution error.** Execution errors occur during program execution after a successful compilation e.g. some common execution errors are the generation of an excessively large numerical quantity, division by zero, or an attempt to compute the logarithm or the square root of a negative number. Diagnostic messages will often be generated in this type of situation, making it easy to identify and correct the errors. These diagnostics are sometimes called execution diagnostics, to distinguish them from the compilation diagnostics.

Interpreters operate differently than compilers. Interpreters can detect both compilation-type errors and execution-type errors line by line, approximately at the same time. When an error is detected during the interpretation process, the interpreter will stop and issue an error message, indicating which line was being processed when the error was detected. But interpreters are generally much slower than compilers while executing a program. Interpreters are therefore less suitable for running debugged programs.

INTEXT QUESTIONS

1. Define "top-down" programming?
2. Why are some statement indented within a C program?
3. What is the extension of C program file?
4. What is a syntactic error?

13.5 DETECTION AND CORRECTION OF ERRORS

Syntactic errors and execution errors usually result in the generation of error messages when compiling or executing a program. Error of this type are usually quite easy to find and correct. There are some logical errors that can be very difficult to detect. Since the output resulting from a logically incorrect program may appear to be error free. Logical errors are often hard to find, so in order to find and correct errors of this type is known as **logical debugging.**

To detect errors test a new program with data that will give a known answer. If the correct results are not obtained then the program obviously contains errors. Even if the correct results are obtained, however you cannot be sure that the program is error free, since some errors cause incorrect result only under certain circumstances. Therefore a new program should receive thorough testing before it is considered to be debugged.

Once it has been established that a program contains a logical error, some ingenuity may be required to find the error. Error detection should always begin with a thorough review of each logical group of statements within the program. If the error cannot be found, it sometimes helps to set the program aside for a while. If an error cannot be located simply by inspection, the program should be modified to print out certain intermediate results and then be rerun. This technique is referred to as **tracing**. The source of error will often become evident once these intermediate calculations have been carefully examined. The greater the amount of intermediate output, the more likely the chances of pointing the source of errors. Sometimes an error simply cannot be located.

Some C compilers include a debugger, which is a special program that facilitates the detection of errors in C programs. In particular a debugger allows the execution of a source program to be suspended at designated places, called **break points,** revealing the values assigned to the program variables and array elements at the time execution stops. Some debuggers also allow a program to execute continuously until some specified error condition has occurred. By examining the values assigned to the variables at the break points, it is easier to determine when and where an error originates.

INTEXT QUESTIONS

5. What is logical debugging?
6. Define tracing.
7. What is a debugger?

13.6 WHAT YOU HAVE LEARNT

In this lesson you learnt how to prepare a complete C program, what files should be included in a C program. You are now aware of the fact that how to compile and execute the C program after writing it through some editor. At last logical and syntactical errors and technique to detect and correct them are also discussed for the benefit of the learners.

13.7 TERMINAL QUESTIONS

1. Explain the advantages of "top-down" programming.
2. Define "bottom-up" programming.
3. What is the difference between a line-oriented editor and a character oriented editor?
4. What is the difference between compilation and execution of a C program?
5. Describe the concept of linking.
6. What are diagnostic message?

13.8 KEY TO INTEXT QUESTIONS

1. When the overall program strategy has been clearly established, then the syntactic details of the language can be considered. Such an approach is often referred to as "top-down" programming.
2. The use of indentation is closely related to the sequencing of groups of statements within a program. Indentation explains the subordinate nature of individual statements within a group.
3. 'C'
4. If some variables are improperly declared, a reference to an undeclared variable or incorrect punctuation then such errors are referred to as syntactical errors.
5. To find and correct errors of logical type is known as logical debugging.
6. If an error cannot be located simply by inspection, the program should be modified to print out certain intermediate results and then be rerun. This is known as tracing.
7. Debugger is a special program that facilitates the detection of errors in other C programs. It allows the execution of a source program to be suspended at designated places, called break point.