Chapter 8

GETTING STARTED WITH C

8.1 OVERVIEW

A computer is a device that follows the instructions given to it. A well-defined set of instructions given to the computer is called a *computer program.* A computer program is written in a programming language. Since the emergence of computer, many programming languages have been developed but the effect of C on the computer world is everlasting. This book will remain incomplete without describing the history of the C. That's why before going into detail; let us have an overview of the history of C.

History of C

The **C** programming language was developed by Dennis Ritchie in 1972 at AT & T Bell Laboratories. It was derived from an earlier programming language named *B*. The B was developed by Ken Thompson in 1969-70 and provided the basis for the development of C. The C was originally designed to write system programs under UNIX® operating system. But over the years its power and flexibility have made it popular in industry for a vide range of applications. The earlier version of C was known as K&R (Kernighan and Ritchie) C. As the language further developed, the ANSI (American National Standards Institute) developed a standard version of the language known as ANSI C.

8.2 DEVELOPING A C PROGRAM (A STEPWISE APPROACH)

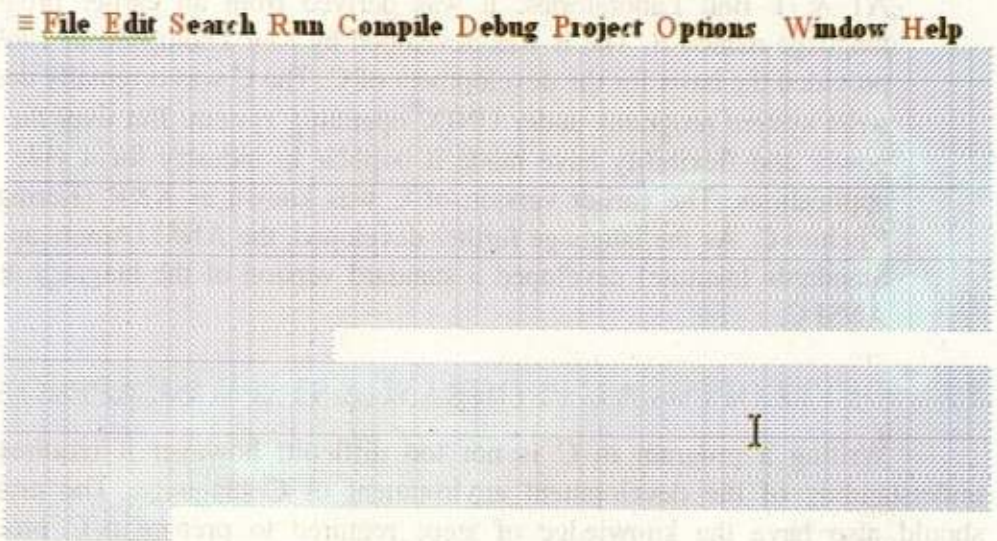
Writing a program in C is not too difficult; however it requires a good understanding of the development environment of C language. The programmer should also have the knowledge of steps required to prepare a C program for execution.

As a first step, install a compiler for the C language on the computer so that the source program can be compiled and executed. Many compilers for C language are available from number of vendors. Any of them can be used, but we recommend using Turbo C++.

8.2.1 Turbo C++ (A Compiler for the C language)

Turbo C++ is a Borland International's implementation of a compiler for C language. In addition to a compiler, TC provides a complete IDE (**I**ntegrated **D**evelopment **E**nvironment) to create, edit and save programs is called TC editor (Fig. 8.1). It also provides a powerful debugger that helps in detecting and removing errors in the program.

Once the TC (Turbo C) has been installed, it is very easy to write C programs in its editor. The IDE can be invoked by typing **tc** on the DOS prompt or by double clicking the TC shortcut. The menu bar of the IDE contains menus to create, edit, compile, execute (Run) and debug a C program. A menu can be opened by either clicking the mouse on it or pressing the first highlighted character of the name of the menu in conjunction with the *Alt* key. For example to open *File* menu, press *Alt+F* (hold down Alt key and then press F key).



8.2.2 Creating and Editing a C Program

To write the first C program, open the *edit* window of the Turbo C++ IDE. This can be done by selecting *File|New* option from the menu bar. A window

appears on the screen (Fig. 8.2). This window has a double-lined border, and a cursor inside the window represents the starting point to write a program.

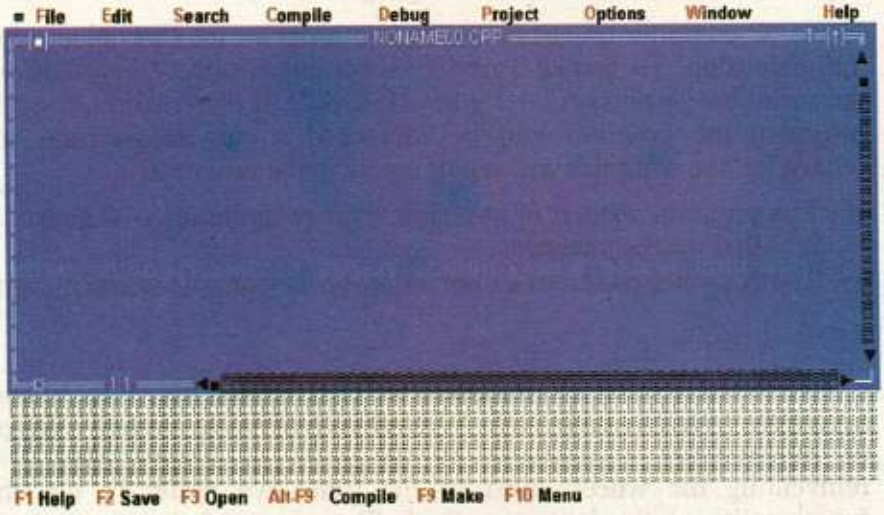


Fig. 8.2 Create, Edit and Save a Program

We can expand this window by clicking the arrow in the upper right corner, or by selecting *Window\Zoom* from the menu bar. We can also navigate through the program by using the vertical and horizontal scroll bars or by using arrow keys.

8.2.3 Saving a C Program

After writing the C program, we should save it on the disk. This can be done by selecting *File\Save* command from the menu bar or pressing the *F2* key. When we select *File\Save*, a dialogue box will appear. At the top of this dialogue box, there is a text box with caption *Save File As*. Type the name of the file in it and press the Enter key. The default path for saving the file is BIN folder. The TC assigns a default name NONAME00.cpp to the file (Fig. 8.2). To save the file in a specific folder / location with a different file name, one has to specify the absolute path.

Note:

Turbo C++ is a compiler for C++ programming language – an extension to C. Therefore it can compile programs of both C and C++. When we save a program with .cpp extension, it can use many additional features that are not supported in ANSI C. *As this course is designed just for C, not C++, therefore it is suggested to always save the programs with .c extension.* When a program is saved with .c extension, the Turbo C++ compiler restricts it to only use standard features of C.

8.2.4 Compiling a C Program

The computer does not understand source program because instructions in the program are meaningless to the microprocessor, as it

understands only the machine language. A program that is to be executed must be in the form of machine language.

C compiler translates the source program into an object program with .obj extension. To invoke Turbo C++ compiler, select *Compile\Compile* from the menu bar or press *Alt + F9* key (Fig. 8.2). If there is no error in the source program, the program will be translated to object program successfully otherwise, the compiler will report errors in the program.

- The program written in any high level programming language, such as C, is called *source program.*
- The compiler produces an *object program* from the source program

8.2.5 Linking a C Program

While writing a C program, the programmer may refer to many files to accomplish various tasks such as input/output etc. In case of C language, a lot of functionality is available in the form of library files. Rather than reinventing the wheel, most of the times we prefer to use the built-in functionality of the language. Such files are needed to be linked with the object file, produced by the compiler, before execution of the program.

Linking is the process in which the object file produced by the compiler is linked to many other library files by the linker. The *linker* is a program that combines the object program with additional object files that may be needed for the program to execute and save the final machine language program as an executable file on disk. In Turbo C++, the linker can be invoked by selecting *Compile\Link* from the menu bar.

The *Linker* combines different library files to the object file and produces an executable file with .exe extension

8.2.6 Executing a C program

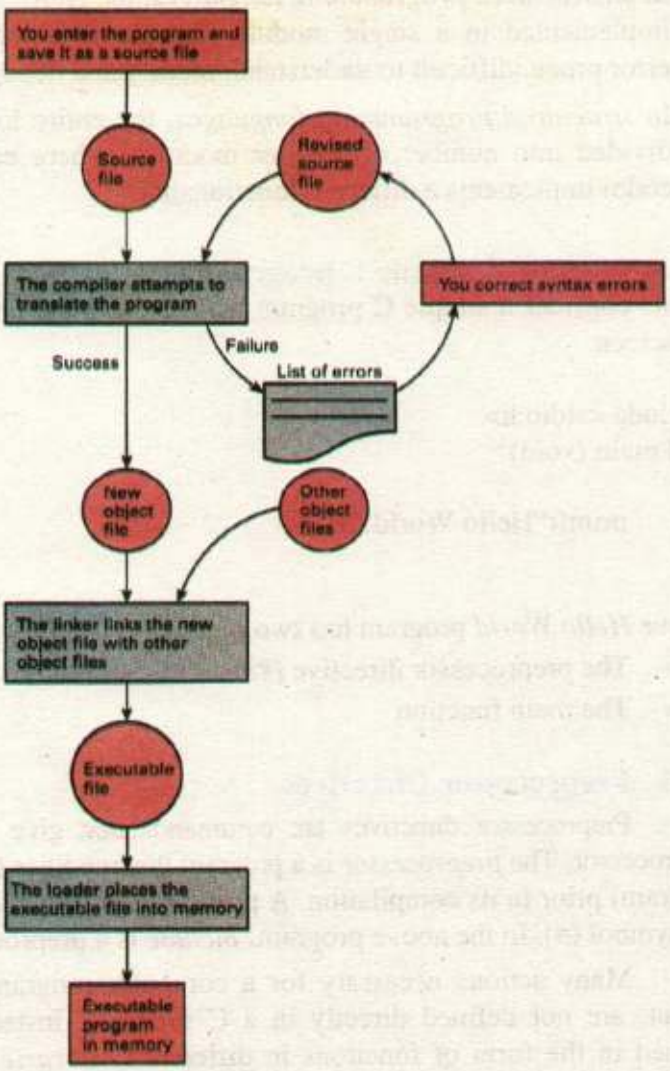
After successfully compiling and linking the program, we are now ready to execute it. For execution the program must be loaded into memory. This is done by the loader. Loader is a program that places executable file in memory. In Turbo C++, this is done by selecting *Run\Run* from the menu bar or pressing *Ctrl+F9* key.

When the program is run, the screen flickers for a moment and the output screen will disappear in a flash. To see the program's output select *Window\User Screen* or press *Alt+F5*. The normal DOS output screen will appear. Flowchart 8.3 describes the steps required to prepare a C program for execution.

Setting the Output and Source Directories

By default, Turbo C++ places the object and executable files in the BIN subdirectory of the TC directory. This is not the right place to put these files. These files should be placed in the same directory where the source file (with .c extension) was created. To do so, select the *Option\Directories* from the

menu bar. A window appears with four fields captioned *Include Directories, Library Directories, Output Directories* and *Source Directories*. The Include Directories filed should already be set to drive:\TC\INCLUDE and Library Directories should be set to drive:\TC\LIB, where the *drive*: is the drive in which the directory TC is placed. It can be C, D, or E etc. We need to set the *output directory* field to source file directory e.g., D:\MyPrograms etc. this is where the compiler will put .obj file and the linker will put .exe file.



8.3 BASIC STRUCTURE OF A C PROGRAM

The structure of a C program is very flexible which increases the power of the language. C is a structured programming language; therefore it provides a well-defined way of writing programs. As discussed earlier in this chapter that a C program is combined with many other files before execution. The linker does this job. But we have to specify these files to be linked. How can this be done? To answer this question and to understand the basic structure of the C program, we proceed with the following example:

- In *unstructured programming languages*, the entire logic of the program is implemented in a single module (function), which causes the program error prone, difficult to understand, modify and debug.
- In *structured programming languages*, the entire logic of the program is divided into number of smaller modules, where each module (piece of code) implements a different functionality.

Hello World – A simple C program

Let us consider a simple C program that displays the phrase *Hello World*! on the screen.

```
#include <stdio.h>
void main (void)
{
        printf("Hello World!");
}
```

The above *Hello World* program has two parts:

- The preprocessor directive (#include <stdio.h>)
- The main function

8.3.1 Preprocessor Directives

Preprocessor directives are commands that give instructions to the C preprocessor. The *preprocessor* is a program that modifies the C program (source program) prior to its compilation. A preprocessor directive always begins with the symbol (#). In the above program, *include* is a preprocessor directive.

Many actions necessary for a computer program, such as input and output, are not defined directly in a C program. Instead, these actions are defined in the form of functions in different C *libraries*. Each library has a standard header file, which is referred to with *.h* extension. In the above program, the *stdio.h* refers to the header file containing the definition of standard input/output functions.

The *include* directive gives a program access to a library. This directive causes the preprocessor to insert definitions from a standard header file into a program before compilation. Hence, the statement **#include<stdio.h>** gives the program access to standard input and output functions.

 # include Directive for Defining Identifiers from Standard Libraries

SYNTAX: #include<*standard header file*>

EXAMPLE: #include <stdio.h>
#include <math.h>

The *include* directive tells the compiler where to find the meanings of standard identifiers (e.g., *printf* in the Hello World program) used in the program. These meanings are described in files called *standard header files*. The header file *stdio.h* contains information about standard input and output functions such as *scanf* and *printf*, whereas the header file *math.h* contains information about common mathematical functions.

Another important preprocessor directive is #define directive. It is used to define a constant *macro*. Examples of this macro will be discussed in subsequent chapters.

#define Directive for Defining Constant Macros

SYNTAX: #define Macro_Name *expression*

EXAMPLE:
```
#define   PI          3.142857
#define   SEC_PER_HR  3600
```

The *expression* may be constant, arithmetic expression or a string. C preprocessor replaces each occurrence of the identifier *Macro_Name* with *value of expression*. The expression of the identifier *Macro_Name* can not be changed during the program execution.

Constant Macro is a name that is replaced by a particular constant value before the program is sent to the compiler.

8.3.2 FUNCTION main

As shown in the above *Hello World* program, the definition of the *main* function comes next to the specification of the *#include* preprocessor directive. In fact, *main* is the function where the execution of the C program begins. Every C program has a *main* function. The rest of the lines of program forms the *body* of the main function, the body is enclosed in braces { and }.

C programs are divided into units called functions. This division is usually done on the basis of functionality, where every function carries out a single task. However, it is no necessary to divide every program into functions. The same functionality may be achieved through a single function. But, every C program must have the functions *main* as the execution of the program starts from there. In this way we can say that the main function is actually the entry point of the C programs.

main Function Definition

SYNTAX:

```
void main (void)
{
        body of main function
}
```

As we know from the algebra that every function returns a single value and may accept one or more arguments (parameters). There is some resemblance between an algebraic function and the *main* function. The definition of the function *main* starts with a reserved word *void*. This *void* represents the data type of the value that is returned by the function, which means the function main returns nothing. The second **void** enclosed in parenthesis describes that the function *main* does not accept any argument. However arguments can be passed to the main function and it can also return a value. But the discussion of this issue is out of scope of this book. You may find the topic in detail in many other books.

Body of the function (enclosed in braces) consists of C language statements, which are used to implement the program logic. There are many types of C statements that help programmers to write C programs. We shall learn much more about writing programs in C in next chapters.

8.3.3 Delimiters

Next to the function definition are braces, which indicate the beginning and end of the function body. These braces are called *delimiters*. The opening brace { indicates the beginning of a block of code (set of statements) while the closing brace } represents the end of a block of code.

8.3.4 Statement Terminator

Every statement in a C program terminates with a *semicolon* (;). If any of the statement is missing the statement terminator, the compiler will report it the following error message.

```
Statement missing ;
```

Note: Always be careful about the semicolon while writing C program statement

8.3.5 Function printf

The last statement in the *Hello World* program is *printf* function. It is used to display the output of the program on the screen. See detail in chapter 3.

8.4 COMMON PROGRAMMING ERRORS

The programmer may come across errors while writing a computer program. In programming languages, these errors are called "bugs", and the processing of finding and removing these bugs is called *debugging*.

When the C compiler detects an error, it displays an error message describing the cause of the error. There are three types of programming errors, these are: Syntax error, Runtime error, and Logic error.

8.4.1 Syntax Errors

A syntax error occurs when the program violates one or more grammar rules of C language. The compiler detects these errors as it attempts to translate the program. If a C statement has syntax error, it can not be translated and the program could not be executed.

There can be many causes of syntax errors, for example, missing statement terminator i.e., the semicolon, using a variable without declaration, missing any of the delimiters i.e., { or } etc.

8.4.2 Runtime Errors

A runtime error occurs when the program directs the computer to perform an illegal operation, such as dividing a number by zero. Runtime errors are detected and displayed by the computer during the execution of a program. When a runtime error occurs, the computer stops executing the program and displays a diagnostic message.

8.4.3 Logical Errors

Logical errors occur when a program follows a faulty algorithm. The compiler can not detect logical errors; therefore no error message is reported

from the compiler. Moreover, these errors don't cause the program to be crashed, that's why these are very difficult to detect. One can recognize logical errors by just looking at the wrong output of the program. Logical errors can only be detected by thorough testing of the program.

8.5 PROGRAMMING LANGUAGES

Programming languages are used to write computer programs. There are two broad categories of programming languages i.e., low level programming languages and high level programming languages. We discuss them briefly to have an overview of both:

8.5.1 Low Level Languages

Low level languages are divided into two broad categories i.e., machine language and assembly language. *Machine language* is the native language of the computer. The computer does not need any translator to understand this language. Programs written in any other language must be converted to machine language so that the computer can understand them. Every machine language instruction consists of strings of binary 0s and 1s. As it is very difficult for human beings to remember long sequences of 0s and 1s, therefore writing programs in machine language are very difficult and error prone. So, it was thought to replace the long sequences of 0s and 1s in machine language with English like word. This idea provided the basis for the development of assembly language.

In *assembly language*, machine language instructions (long sequences of 0s and 1s) are replaced with English like words known as mnemonics (pronounced as Ne-Monics). An assembler (language translator for assembly language programs) is used to translate an assembly language programs into machine language.

8.5.2 High Level Languages

Programming languages whose instructions resemble the English language are called *high level languages*. Every high level language defines a set of rules for writing programs called *syntax* of the language. Every instruction in the high level language must confirm to its syntax. If there is a syntax error in the program, it is reported by the language translator (compiler or interpreter). The program does not translate into machine language unless the error is removed.

Common high-level languages include C, C++, Java, Pascal, FORTRAN, BASIC, and COBOL etc. Although each of these languages was

designed for a specific purpose; all are used to write variety of *application software*. Some of these languages such as C and C++ are used to write *system software* as well. Each of these languages has some advantages and disadvantages over the other e.g., FORTRAN has very powerful mathematical capabilities while the COBOL is ideal for writing business applications, C and C++ are very handy for writing system software while Java is equipped with strong network programming features. Besides having different features, all high level programming languages have some common characteristics are:

- These are English like languages, hence are close to human languages and far from the machine language and are very easy to learn
- Programs written in high level languages are easy to modify and debug, and more readable
- These languages let the Programmers concentrate on problem being solved rather than human-machine interaction
- These describe a well defined way of writing programs
- These do not require a deep understanding of the machine architecture
- High level languages provide machine independence. It means programs written in a high level language can be executed on many different types of computers with a little modification. For example, programs written in C can be executed on Intel® processors as well as Motorola processors with a little modification.

Exercise 8c

1 Fill in the blanks:

(i) The set of instruction given to the computer to solve any kind of problem is called _____

(ii) ANSI stands for _____

(iii) The program written in high level language is known as _____

(iv) _____ is a program that places the executable file in memory

(v) In _____programming language , the entire logic of the program is implemented in a single module

(vi) _____are command that give instruction to C preprocessor

(vii) _____ is a name that is replaced by particular constant before program is sent to the compiler

(viii) The _____ directive gives a program access to a library file

(ix) C program is divided into units, called _____

(x) Every statement in a C program terminates with a _____

(xi) A language translator for Assembly language is called _____

(xii) A set of rule for writing program in high level language is known as_____

2 Choose the correct option:

(i) C is a:

a) High Level Language b) Low Level Language
c) Assembly Language d) Machine Language

(ii) Turbo C++ can compile:

a) C++ programs only b) C and C++ programs
c) Turbo C programs only d) Turbo C++ programs only

(iii) Debug is the process of:

a) Creating bugs in program b) Identifying and removing errors
c) Identifying Errors d) Removing Errors

(iv) C was designed to write programs for:

a) Windows operating system b) Solaris operating system
c) Unix operating system d) OS/2 operating system

(v) Preprocessor directives are commands for:

a) Microprocessor b) Language processor
c) C preprocessor d) Loader

(vi) The expression in define directive:
a) can only be changed at the end of the program
b) can not be changed
c) can not be changed but can be redefine
d) can not be assigned a value

(vii) Which of the following language requires no translator to execute the program:
a) C
b) C++
c) Machine language
d) Assembly language

(viii) .exe file is produced by the:
a) Linker
b) Loader
c) Compiler
d) Interpreter

(ix) Which of the following key is used to save a file?
a) F2
b) F3
c) F5
d) F9

(x) void occupy how many bytes in memory?
a) zero
b) one
c) two
d) four

3 Write T for true and F for false statement:

(i) The C programming language was developed by Dennis Ritchie in 1972.

(ii) C was derived from earlier programming language named B.

(iii) The B was developed by Ken Thomson in 1980.

(iv) The short-key for compiling a program is Alt+F9.

(v) The compiler produces the source file from an object file.

(vi) The linker is a program that combines the object program with additional files.

(vii) The short-key for executing the C program is Alt + F5.

(viii) In structured programming the entire program is divided into smaller modules.

(ix) The value of a constant can be changed during the program execution.

(x) High level language provide machine independence.

4 Briefly describe the history of C.

5 List two reasons why it would be preferable to write a program in C rather than machine language.

6 What necessary steps taken to prepare a C program for execution? Explain with diagram.

7 Define a *bug*. Discuss some debugging features of Turbo C++.

8 While writing a C program, how many types of errors can occur? Discuss briefly. Which one is the most difficult to locate and remove? Justify your answer.

9 What is a programming language? Discuss the two main categories of programming languages.

10 Describe characteristics of high-level programming languages.

11 Briefly describe the basic structure of a C program.

12 How would you create, edit, compile, link and execute a C program? Discuss briefly.

13 Differentiate the following:

(i) Preprocessor Directive and the Compiler

(ii) Structured and Unstructured programming languages

(iii) Linker and Loader