Chapter # 08
Getting Started with C

Q. What is program and programming language? Discuss two main categories of programming languages.

Program

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. A person who develops program is called **programmer**.

Programming Language

A set of words and symbols used to write programs is called **programming language**. The programming languages are used to write computer programs. A programming language is a means of communication between a user and computer.

Type of Programming Languages

There are two types of computer programming languages:

1. Low-level languages
2. High-level languages

1. Low Level Languages

Low-level languages are near to computer hardware and far from human languages. The low-level languages are divided into the following two main categories:

- Machine Language
- Assembly Language

a) Machine Language

A type of language in which instructions are written in binary form is called **machine language**. It is the only language that is directly understood by the computer. It is the native language of the computer.

A program written in machine language can be executed very fast by computer. The computer does not need any translator to understand this language. The programs written in machine language are **machine-dependent**. Every computer has its own machine language.

Machine language is difficult to understand. The process of writing and modifying program in machine language takes a lot of time.

b) Assembly Language

Assembly language is a low-level language. It is one step higher than machine language. In assembly language, machine instructions are replaced with English-like words known as **mnemonics**. It is pronounced as **Ne-Monics**.

Programs written in assembly language are easier to write and modify than machine language. Assembly language is mostly used for writing system software. A translator called assembler is used to convert assembly language programs into machine language.

2. High Level Languages

A type of language that is close to human languages is called **high-level language**. The instructions in these languages are similar to English language such as **input** and **print** etc. These languages are easy to understand.

Every high level language defines a set of rules for writing programs called **syntax**. Each instruction must be written according to the syntax of the language. Any error in the program is indicated by the language translator. A program cannot be converted into machine language if it contains any syntax error.

Commonly Used High-level Languages

Common high-level languages include the following:

- **C/C++:** It is used to write system software and application software.
- **Java:** It provides strong features for network programming.
- **Pascal:** It is used for both scientific and business applications.
- **FORTRAN:** FORTRAN stands for FORmula TRANslation. It has very powerful mathematical capabilities.
- **BASIC:** BASIC stands for Beginner All Purpose Symbolic Instruction Code. It was used mainly by students to use the computer for solving simple problems.
- **COBOL:** COBOL stands for Common Business Oriented Language. It is specially designed for business applications.

Q. Describe the characteristics of high-level programming languages.

Some important characteristics of high-level languages are as follows:

1. Easy to Learn

High-level languages are closer to human languages and far from machine language. These are English-like languages and are easier to learn.

2. Easy Error Detection

High-level languages are easy to read and modify. It makes it easy to find errors in programs written in high-level languages.

3. Standardized Syntax

The syntaxes of high-level languages are standardized. These languages describe a well-defined way of writing programs. Different organizations work to determine standard syntax of these languages. An important organization is American National Standard Institute also known as ANSI.

4. Deep Hardware Knowledge not Required

These languages do not require deep knowledge of hardware or machine architecture. A programmer can write efficient programs without a deep knowledge of hardware. He can concentrate on solving the problem rather than concerning the human-machine interaction.

5. Machine Independence

High-level languages provide machine independence. It means that the programs written in high-level language can be executed on different types of computers. For example, a program written in C can be executed on Intel processors and Motorola processors.

6. More Programmers

Programming in low-level languages is very difficult. But high-level languages are easy to learn. It encourages more people to learn these languages. So many programmers of high-level languages are available.

7. Shorter Programs

Programs written in high-level languages are shorter than low-level languages. One instruction of high-level language is equivalent to many instructions of low-level language.

Q. Differentiate between low-level and high-level languages.

The main difference between low-level and high-level program is as follows:

High-level Language	Low-level Language
1. High-level languages are easy to learn.	1. Low-level languages are difficult to learn.
2. These are near to human languages.	2. These are far from human languages.
3. Programs in high-level languages are slow in execution.	3. Programs in low-level languages are fast in execution.
4. Programs in high-level languages are easy to modify.	4. Programs in low-level languages are difficult to modify.
5. High-level languages do not provide much facility at hardware level.	5. Low-level languages provide facility to write programs at hardware level.
6. Deep knowledge of hardware is not required to write programs.	6. Deep knowledge of hardware is required to write programs.
7. These languages are normally used to write application software.	7. These languages are normally used to write system software.
8. There are many programmers of high-level languages.	8. There are a few programmers of low-level languages.

Q. Define source code and object code. What is difference between them?

Source Code

A program written in a high-level language is called **source code**. Source code is also called **source program**. Computer cannot understand the statements of high-level language. The source code cannot be executed by computer directly. It is converted into object code and then executed.

Object Code

A program in machine language is called **object code**. It is also called **object program** or **machine code**. Computer understands object code directly.

Difference between Source Code and Object Code

The main difference between source code and object code is as follows:

Source Code	Object Code
1. Source code is written in high-level or assembly language:	1. Object code is written in machine language through compilers.
2. Source code is easy to understand.	2. Object code is difficult to understand.
3. Source code is easy to modify.	3. Object code is difficult to modify.
4. Source code contains fewer statements than object code.	4. Object code contains more statements than source code.

Q. Describe language processors or translators and their use. Discuss different types of language processors.

Computer understands only machine language. A program written in high-level or assembly language cannot be run on a computer directly. It must be converted into machine language before execution. Language processor or translator is a type of system software that converts these programs into machine language. Every computer language has its own translators.

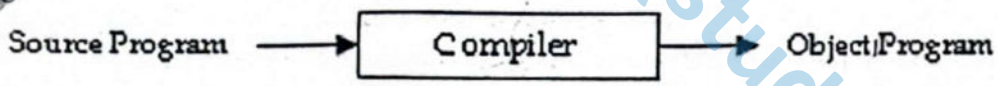
Types of Translators

Different types of translators or language processors are as follows:

- Compiler
- Interpreter
- Assembler

1. Compiler

A compiler is a program that converts the instruction of a high level language into machine language as a whole. A program written in high-level language is called source program. The compiler converts the source program into machine code. The machine code is known as object program. The object program can be executed many times.



The compiler checks each statement in the source program and generates machine instructions. Compiler also checks syntax errors in program. A source program containing an error cannot be compiled.

A compiler can translate the programs of only those languages for which it is written. For example, C compiler can translate only those programs that are written in C language.

2. Interpreter

An interpreter is a program that converts one statement of a program into machine at one time. It executes a statement before translating the next statement of source program. If there is an error in the statements, interpreter stops working and displays an errors message.

The advantage of interpreters over compilers is that an error is found immediately. So the programmer can correct errors during program development.

The disadvantage of interpreter is that it is not very efficient. The interpreter does not produce an object program. It must convert the program each time it is executed. Visual Basic uses interpreter.

3. Assembler

An assembler is a translating program that translates the instruction of assembly language into machine language.

Q. What is difference between compiler and interpreter?

Following is a brief difference between compiler and interpreter:

Compiler	Interpreter
1. Compiler converts a program into machine code as a whole.	1. Interpreter converts a program into machine code statement by statement.
2. Compiler creates object code file.	2. Interpreter does not create object code file.
3. Compiler converts high-level program that can be executed many times.	3. Interpreter converts high-level program each time it is executed.
4. Program execution is fast.	4. Program execution is slow.
5. Compiler displays syntax errors after compiling the whole program.	5. Interpreter displays the syntax error on each statement of program.

Q. Briefly describe the history of C language.

C is a popular high-level language. It was developed by **Dennis Ritchie** at **AT&T Bell Laboratories** in 1972. C language was derived from an earlier programming language called **B**. The **B** was developed by **Ken Thompson** in 1969-70. It provided the basis for the development of C. The C language was originally designed to write system programs under **UNIX** operating system. The power and flexibility of C language made it popular in industry for a wide range of applications.

The earlier version of C was known as **K&R** (Kernighan and Ritchie) C. The **American National Standard Institute (ANSI)** developed a standard version of the language. The standard version is known as **ANSI C**. This new version provided many features that were not available in the older version.

Q. Write some advantages or characteristics of C language.

Some important advantages of C language are as follows:

1. Convenient Language

C is very convenient language. It provides many facilities in easier way that are *difficult* to use in *low-level languages*. Programmers can write complex programs more easily as compared to low-level languages.

2. Well-Structured Language

C is a well-structured language. Its syntax is very easy to understand. The programs written in C language are easy to maintain and modify.

3. Machine Independence

C language provides machine independence. It means that the programs written in C language can be executed on different types of computers. For example, a program written in C can be executed on Intel processors and Motorola processors. That is why it is preferable to write program in C rather than machine language.

4. Modularity

C language provides the facility of modular programming. It means that the program can be divided into small modules. These modules can be developed and compiled independently.

5. Case Sensitivity

C is a case sensitive language. It means that it can differentiate uppercase and lowercase words. All keywords are written in lowercase. This feature makes it easier to maintain the source code.

6. Hardware Control

C language provides close control on hardware. It can be used to write efficient programs to control hardware components of computer system.

7. Small Language

C is a small language. It has a small number of keywords and programming controls. But still it is very powerful for developing different types of programs.

8. Fast Code Generation

The compilers of C language generate very fast code. The code executes very efficiently. So the programs take less time to execute.

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

The two reasons to write a program in C rather than machine language are as follows:

1. Easier to Write Program

C language is a high-level language. It is easier to write programs in C than machine language. The programs in machine language consist of 0 and 1. It is very difficult to write, modify and manage programs in machine language. The instructions of C are similar to English language. So it is preferable to write program in C rather than machine language.

2. Machine Independence

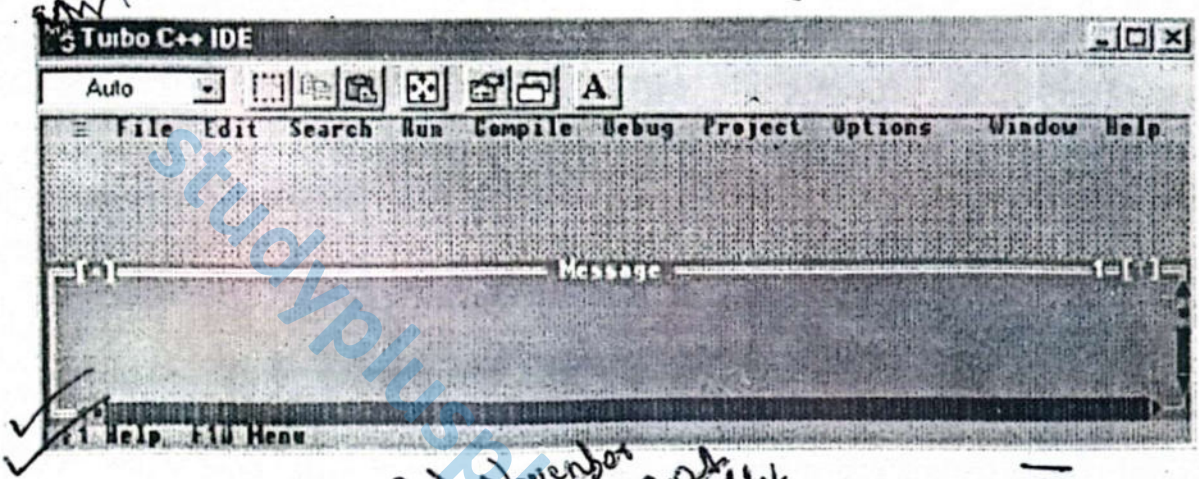
C language provides machine independence. It means that the programs written in C language can be executed on different types of computers.

For example, a program written in C can be executed on Intel processors and Motorola processors. That is why it is preferable to write program in C rather than machine language.

Q. Write a note on Turbo C++.

The compiler used for C language is called **Turbo C++**. It is the implementation of Borland International for C language. Turbo C++ also provides a complete **Integrated Development Environment (IDE)** known as **TC editor**. It is used to create, edit and save programs. It also provides a powerful debugger. The debugger helps users in detecting and removing errors in programs.

The process of writing programs in Turbo C editor is very easy. The user can type "TC" on DOS prompt or double click TC shortcut to start IDE. The menu bar of IDE contains menus to create, edit, compile, execute and debug a C program. The user can open a menu by clicking on it with the mouse. The user can also press a combination of ALT key and the first highlighted character of the name of menu. For example, **ALT+F** is used to open **File** menu.

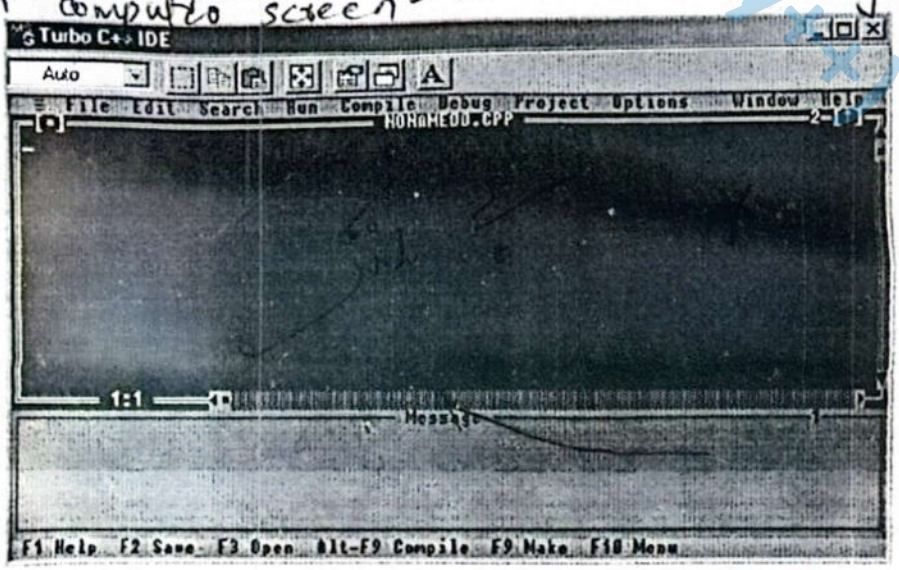


Q. How would you create, edit, compile, link and execute a C program? Discuss briefly.

A process to create, edit, compile, link and execute a C program is as follows:

Creating and Editing a C Program

The process of writing C program is known as **editing**. This process includes writing, modifying and deleting program statements. The part of Turbo C++ IDE that is used to write C programs is called **edit window**. A new edit window can be opened by selecting **File > New** option from menu bar. The edit window appears as follows:



The edit window has a double-lined border. The cursor blinks in the window. The position of the cursor indicates the starting point to write a program. The user can expand the window by clicking the arrow in the upper right corner. The user can also select **Window > Zoom** to expand the window. The vertical and horizontal scrollbars are used to navigate through the program.

Saving a C Program

The process of storing the program on disk is known as **saving**. A program should be saved on disk to be used repeatedly. C programs are saved with **.c** extension. For example, a program can be saved like **program.c**. The following procedure is used to save a C program:

1. Select **File > Save** OR press **F2** key. The **Save File As** dialog box will appear. The default name **NONAME00.CPP** appears in the dialog box.

2. Enter the file name.
3. Enter the path to save file. The default location to save programs is **BIN** directory.
4. Click **OK**. The program will be saved at specified location with specified name.

Compiling a C Program

The process of converting source program into object program is known as **compiling**. The program saved with .c extension contains the statement of C language. It is known as source program. The source program cannot be executed by computer directly. A compiler converts the source program into object program and saves it in a separate file. The object program is saved with .obj extension.

The source program cannot be compiled if it contains any syntax error. The compiler generates error message to describe the cause of error. All errors must be removed to successfully compile a source program.

The following procedure is used to compile a C program:

Select **Compile > Compile** OR press **ALT+F9** key. The program will be translated into object program if it contains no error. The compiler will generate error message if the program contains any error.

Linking a C Program

The process of linking library files with object program is known as **linking**. These files are used to accomplish different tasks such as input/output. A **library file** must be linked with the object file before execution of program. A program that combines the object program with additional library files is known as linker. It is part of C++ compiler.

The linker generates error message if the library file does not exist. A new file is created with .exe extension if the process of linking is successful. This file is known as executable file and can be executed by the computer directly. The linker can be invoked in Turbo C++ by selecting Compiler > Link from the menu bar.

Executing a C Program

The process of running an executable file is known as executing. The C++ program can be executed after compiling and linking. The program must be loaded into the memory to execute. A program that places an executable file in the memory is known as loader. The program can be loaded in the memory by selecting Run > Run from menu bar or pressing CTRL+F9.

The screen flickers for some time when a program is executed. The output screen displays the output of the program and disappears. The user can display the output screen by selecting Window > User Screen or pressing ALT+F5.

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

Different necessary steps are taken to prepare a C program for execution. These steps are executed in a sequence. Different steps to prepare C program for execution are as follows:

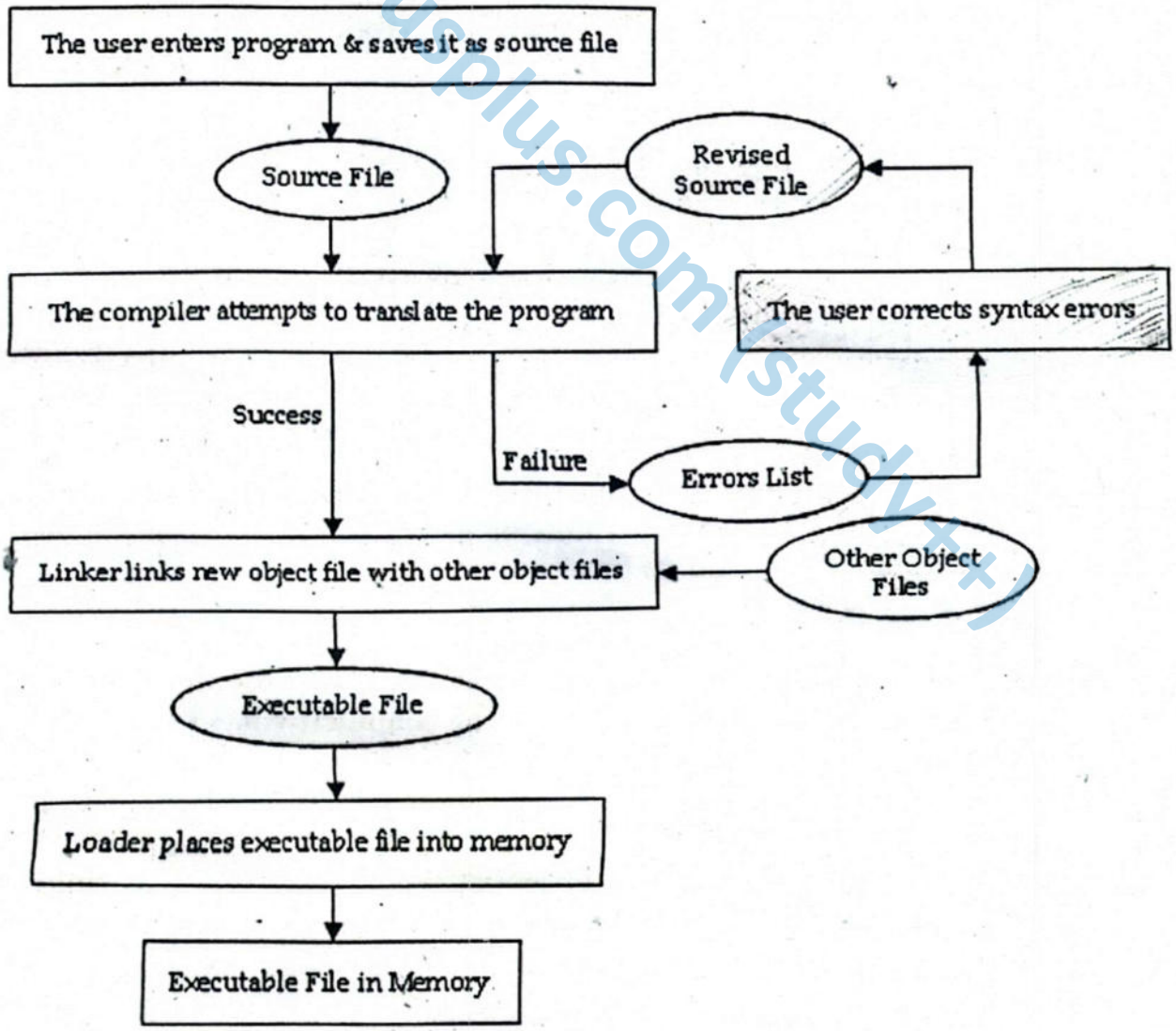


Figure 8.1: Steps to prepare C program for execution

1. Creating and Editing a C Program

The first step is to create and edit a new program. It includes writing, modifying and deleting program statements. This process is performed by using text editor like Note Pad, WordPad etc. Most compilers provide a text editor for writing source code of C programs.

2. Saving a C Program

The process of storing the program on disk is known as **saving**. A program should be saved on disk to be used repeatedly. The C program is saved with **.c** extension.

3. Compiling a C Program

The process of converting source program into object program is known as **compiling**. The program saved with **.c** extension contains the statement of C language. It is known as **source program**. The source program cannot be executed by computer directly. A compiler converts the source program into object program and saves it in a separate file. The object program is saved with **.obj** extension.

The source program cannot be compiled if it contains any syntax error. The compiler generates error message to describe the cause of error. All errors must be removed to successfully compile a source program.

4. Linking a C Program

The process of linking library files with object program is known as **linking**. These files are used to accomplish different tasks such as input/output. A **library file** must be linked with the object file before execution of program. A program that combines the object program with additional library files is known as **linker**. It is part of C++ compiler.

The linker generates error message if the library file does not exist. A new file is created with **.exe** extension if the process of linking is successful. This file is known as executable file and can be executed by the computer directly.

5. Loading a C Program

In this step, the loader loads the executable file into the memory for execution. A program must be loaded in the memory in order to execute it. A program that places an executable file in the memory is known as **loader**.

6. Executing a C Program

The process of running an executable file is known as **executing**. The C++ program can be executed after compiling and linking. In this last step, the program is executed. The instructions written in the program are executed by the computer.

Q. Differentiate between linker and loader.

A program that combines the object program with additional library files is known as **linker**. It is part of C++ compiler. It is used to perform the process of **linking**. In this process, the library files are linked with object program. These files are used to accomplish different tasks such as input/output etc. The linker generates error message if the library file does not exist. A new file is created with .exe extension if the process of linking is successful. This file is known as executable file and can be executed by the computer directly.

A program that places an executable file in the memory is known as **loader**. A program must be loaded in the memory in order to execute it.

Q. What is the difference between ".c" and ".cpp" extensions?

Turbo C++ is a compiler for C++ programming language. C++ is an extension of C language. Therefore, the compiler can compile the programs of C++ as well as C.

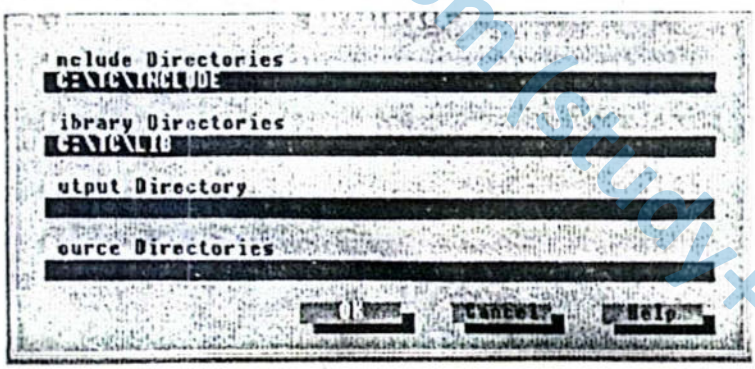
The compiler can use many additional features if the source program is saved with .cpp extension. These features are not supported in ANSI C. The compiler restricts a program to only standard features of C if the source program is saved with .c extension.

Q. Briefly describe the process of setting the output and source directories.

By default, **Turbo C++** stores object files and executable files in **BIN** subdirectory of **TC** directory. It is not the right place for storing these files. These files should be stored in the same directory in which source files are stored. The user can change the setting for output and source directories. The following procedure is used to set output and source directories:

1. Select **Options > Directories**. A window will appear with four fields.

- **Include Directories:** It indicates the location of standard header files included in C programs. It should already be set to *drive*:\TC\INCLUDE. The *drive* can be any drive on computer.
- **Library Directories:** It indicates the location of library files. It should already be set to *drive*:\TC\LIB. The *drive* can be any drive on the computer.
- **Output Directories:** It indicates the location where the compiler stores object files and linker stores executable files.
- **Source Directories:** It indicates the location of source code of the libraries that do not belong to the open project.



2. Enter the path for output directory like "C:\MyPrograms" in **Output Directory** field.
3. Enter the path for Source directory like "C:\MyPrograms" in **Output Directory** field.
4. Click **OK**. The new setting will be applied.

Q. Briefly describe the basic structure of a C program.

The format of writing C program is called its **structure**. *The basic structure of a C* program is very flexible. It increases the power of the language. It consists of following parts:

- Preprocessor directive
- Main() function
- Program body (C statements)

Example

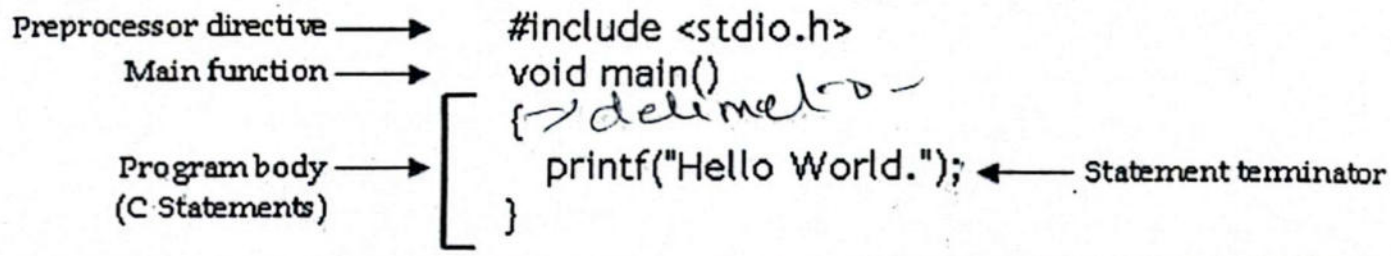


Figure 8.2: Structure of C program

In the above figure,

- First line is preprocessor directive to include a header file **stdio.h**. The **preprocessor directives** are the commands that give instructions to C preprocessor. **Preprocessor** is a program that modifies C source program before compilation. The preprocessor directives start with hash symbol #.
- The second line is **main** function. The **main()** function is the place where execution of a C program starts. When the program is executed, the control enters **main()** function and starts executing its statements.
- The statements of the program are written in curly braces. The curly brace { is called opening brace and } is called closing brace. The braces are also known as **delimiters**. These statements are collectively known as the **body** of a program.
- Every statement in C program is terminated with a semi colon (;). The compiler generates an error if any statement is not terminated by semi colon.

Q. What is preprocessor directive? Explain its purpose.

Preprocessor directive is an instruction given to the compiler before the execution of actual program. Preprocessor directive is also known as **compiler directive**. The preprocessor directives are processed by a program known as **preprocessor**. It is part of C++ compiler. It modifies C source program before compilation. The semicolon is not used at the end of preprocessor directives.

The preprocessor directives start with hash symbol # and the keyword **include** or **define**. These directives are written at the start of program. The preprocessor directive is used in C to include header files in the program.

Q. Discuss different types of preprocessor directives.

Two types of preprocessor directives used in C language are as follows:

1. Include Preprocessor

The **include** preprocessor directive enables a program to access a library. Each library contains different header files. The include preprocessor directive is used to include header files in the program.

Syntax

The syntax of using this directive is as follows:

```
#include <standard header file>
```

Example

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

The above statement tells preprocessor to include the file **stdio.h** and **math.h** in source program before compiling it.

The **include** directive tells the compiler where to find the meanings of standard identifiers such as **printf**. The meanings are described in **standard header files**. The header file **stdio.h** contains information about standard input and output functions such as **scanf** and **printf**. The **math.h** contains information about common mathematical functions.

2. define Preprocessor

The **define** directive is used to define a constant. It starts with the symbol #. It is not terminated with semicolon. It can be used anywhere in the program.

Syntax

The syntax of define directive is as follows:

```
#define  identifier  value
```

#	It indicates the start of preprocessor directive.
define	It is used to define a constant.
identifier	It is the name of the constant.
value	It represents the value associated with the identifier.

The preprocessor directive replaces all occurrences of the **identifier** with the **value**. The identifier is conventionally written in uppercase.

Example

```
#define PI  3.141593
```

Q. Explain header files and their use in programs.

Header files are collection of standard library functions to perform different tasks. There are many header files for different purposes. Each header file contains different types of predefined functions. Many header files can be included in one program. The header file must be included in the program before calling any of its functions in the program.

The extension of a header file is .h. The **include** preprocessor directive is used to include header files in programs. These files are provided by C compiler system.

The header files are normally stored in **INCLUDE** subdirectory. The name of header file is written in angle brackets < >.

Syntax

The syntax of using header files is as follows:

```
#include <header_file_name>
```

The name of header file can also be used in double quotes as follows:

```
#include "header_file_name"
```

Examples

```
#include "stdio.h"
```

The word "stdio" stands for **standard input/output**. This header file contains the definitions of built-in input and output functions.

A header file **math.h** is used in programs to use predefined mathematical functions. The following statement is used to include this file in program:

```
#include "math.h"
```

Q. Explain main function used in C programs.

The **main()** function is the place where the execution of a C program starts. When the program is executed, the control enters **main()** function and starts executing its statements.

Each program must contain **main()** function. If a program does not contain main function, it can be compiled but cannot be executed. Any number of statements can be written in the body of the **main()** function. The body is enclosed in braces {}.

Syntax

The syntax of **main()** function is as follows:

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

The main function consists of the following:

- The definition of main function starts with keyword **void**. It indicates the type of value that is returned by function. void means that the function will return no value.
- Keyword **void** in parenthesis indicates that function does not accept any argument.
- The body of main function is enclosed in braces. It consists of C language statements. The instructions are used to implement program logic.

Q. What do you know about C statements?

A statement in C language is an instruction for the computer to perform a task. The statements are written in curly brackets. Computer performs these instructions one by one in the same sequence in which these instructions are written. Each statement in C is terminated with semicolon(;).

Example

The following example contains two statements in the body of main() function.

```
#include <stdio.h>
void main()
{
    printf("Hello World of C Programming.");
    printf("Programming makes life interesting.");
}
```

Q. Define bug and debugging.

An error in a computer program is known as bug. The programmer can make different errors while writing programs. A program cannot be compiled if it contains any syntax error. The compiler detects errors and displays errors message to describe the cause of error.

The errors must be removed from the program before it can be compiled and executed. The process of finding and removing bugs is known as debugging.

Q. 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.

Different types of errors can occur while writing a C program. These errors include the syntax errors, logical errors and run-time errors.

1. Syntax Errors

A collection of rules for writing programs in a programming language is known as syntax. All program statements are written according to these rules. Syntax error is a type of error that occurs when an invalid statement is written in program. The compiler detects syntax errors and display error message to describe the cause of error. A program containing syntax errors cannot be compiled successfully.

There can be many causes of syntax errors. Some important causes are as follows:

- The statement terminator is missing at the end of statement.
- A misspelled keyword is used in the program.
- Any of the delimiters is missing.

Example

Typing **forr** instead of **for** is an example of syntax error.

2. Logical Errors

A type of error that occurs due to poor logic of the programmer is called **logical error**. A statement with logical error may produce unexpected and wrong results in the program. Logical errors are difficult to find because translator cannot detect these errors. The logical errors can only be detected by examining the program thoroughly.

Examples

Some examples of logical errors are as follows:

- Using wrong conditions in program such as writing a < 5 instead of a > 5.
- Using wrong formula in the program such as writing Average = Total * 5 instead of Average = Total / 5.

3. Run-Time Errors

A type of error that occurs during the execution of program is known as run-time error. It occurs when a statement directs the computer to execute an illegal operation such as dividing a number by zero.

The run-time errors are detected and displayed by the computer during execution. Run-time errors normally occur due to wrong input from the user. The computer stops executing the program and displays error message if a runtime error occurs.

Example

The user may ask the program to open a file that does not exist. Similarly, the user may enter wrong type of data etc.

Most Difficult Error (logical error) –

The most difficult type of error is logical error for the following reasons:

- It cannot be detected by the compiler.
- It does not crash the program. That is why it is difficult to detect.
- The user needs to review the whole program to find out logical error.
- It may take a lot of time for detecting logical error.

Q. Discuss some debugging features of Turbo C++.

Turbo C++ provides many useful debugging features. The debugging features of Turbo C++ are available in Debug menu. Some important debugging features of Turbo C++ are as follows: The process of finding and removing bug is called debugging –

1. Single Stepping

A program may not execute even if it has been compiled successfully. The debugger provides the facility to find errors by executing one line of a program at one time. It enables the programmer to detect the exact place of error. The following procedure is used to execute one step at a time with single stepping:

- Select **Run > Trace Into** OR press **F7**.

2. Watches

The **watch** or **watch expression** is used to check the value of a variable as the program executes. It indicates how the values of variables change during program execution. It is normally used in combination with single stepping.

The following procedure is sued to use watch or watch expression:

- Place the cursor on the variable whose value is to be checked.
- Select **Debug > Watches**. A submenu will appear.
- Select **Add Watch** from the submenu. OR press **CTRL+F7**. A dialog box will appear. The selected variable will appear in **Watch expression** field.
- Click **OK** OR press **Enter**. A new window will appear indicating that the selected symbol is undefined.
- Select **Run > Trace Into** OR press **F7** to execute single stepping. The value of the selected variable will appear in **Watch** window.

3. Breakpoints

A **breakpoint** is used to mark a part of program where program execution will stop. The program executes all statements up to the breakpoint and then stops. The user can check the values of a variable at this point by using **Watch** window by single stepping the remaining part of the program.

The following procedure is used to insert a breakpoint in the program:

- Place the cursor on the line on which the breakpoint is to be inserted.
- Select **Debug > Toggle breakpoint** OR press **CTRL+F8**. The breakpoint will be inserted and the line will be highlighted.

4. Evaluate/Modify Window

The **evaluate/modify** window is used to change the value of variable during program execution. It can be useful if the user is single stepping the program and wants to change the value of a certain variable. The following procedure is used to use evaluate/ modify window:

- Select **Debug > Evaluate/Modify**. A new window will appear with three fields.
- Enter the name of the variable whose value is to be modified in **Expression** field.
- Enter the new value for the variable in **New Value** field. The value of the third field **Result** will also change automatically.

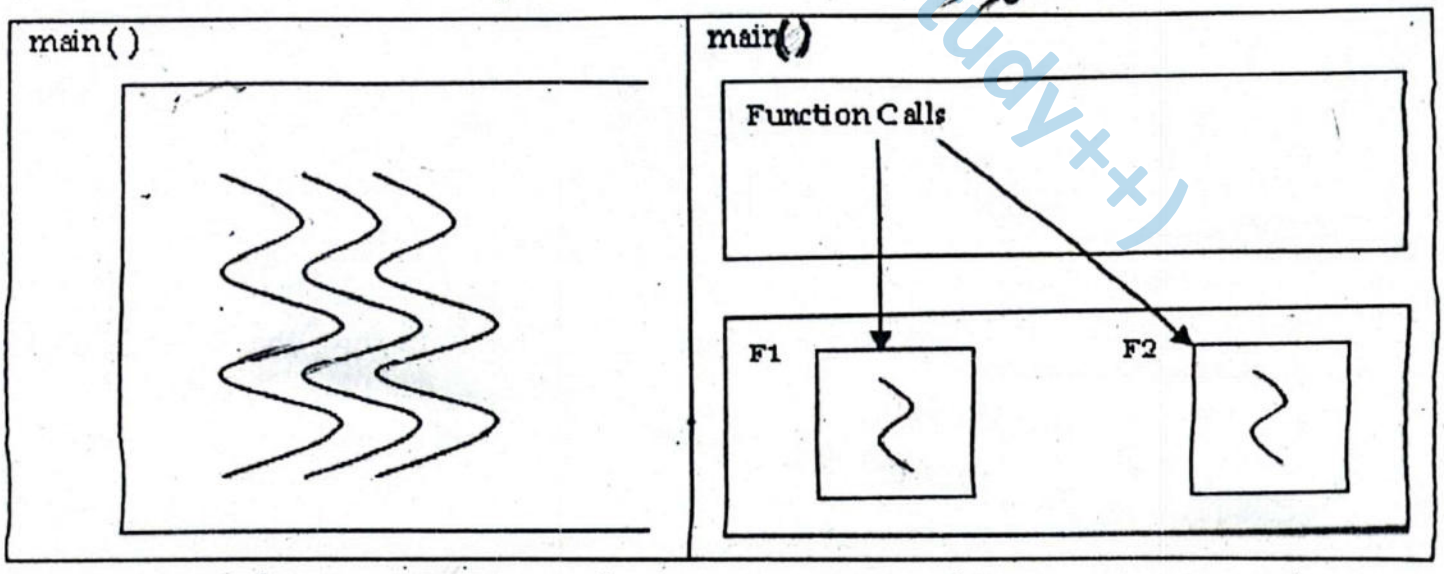
Q. What is difference between unstructured and structured programming languages?

Unstructured Programming Language

In unstructured programming languages, the entire logic of the program is implemented in a single module or function. The programs written in these languages are error prone, difficult to understand, modify and debug.

Structured Programming Language

In structured programming languages, the entire logic of the program is divided into a number of smaller modules or functions. Each module is a piece of code that implements a different functionality. The main module calls other modules when they are needed to execute. It is a modular method of writing programs. It is an easy and simple technique to write program. The programs written in these languages are easy to understand, modify and debug. There is also a less possibility of errors in the programs.



Unstructured Programming Approach Structured Programming Approach

Figure 8.3: Difference between Structured and Unstructured Approaches

Q.1. What is ANSI C?

The American National Standard Institute (ANSI) developed a standard version of the language. The standard version is known as ANSI C.

Q.2. List some advantages or characteristics of C language?

- Convenient and well-structured language
- Machine Independence
- Modularity

Q.3. What do you mean by machine independence?

Machine independence means that programs written in one language can be executed on different types of computers. For example, a program written in C can be executed on Intel processors and Motorola processors.

Q.4. Define program.

A well-defined set of instructions given to the computer is called a computer program. A computer program is written in a programming language.

Q.5. Define programming language.

A set of words and symbols used to write programs is called programming language. The programming languages are used to write computer programs. A programming language is a means of communication between a user and computer.

Q.6. Who develops computer programs?

A person who develops computer programs is called programmer. The programmer develops programs to instruct the computer how to process data to convert into information. Programmer uses programming languages or tools to write programs.

Q.7. Name two main categories of programming languages.

The two main categories of computer programming languages are low-level languages and high-level languages.

Q.8. Why does machine language program execute faster?

A program written in machine language can be executed very fast by computer. The computer does not need any translator to understand this language.

Q.9. Define low level language.

A language that is close to hardware and far from human language is called low level language.

Q.10. Define assembly language.

Assembly language is a low-level language. It is one step higher than machine language. In assembly language, machine instructions are replaced with English-like words known as mnemonics.

Q.11. Define high level language.

A type of language that is close to human languages is called high level language. The instructions in these languages are similar to English language such as input and print etc. These languages are easy to understand.

Q.12. Differentiate between assembly language and high-level language.

In assembly language, one symbolic instruction must be written for each machine language instruction. In high level language, one statement can produce multiple machine

language instructions. Another difference is that assembly language is closely tied to a computer's machine language. It means that it is impossible to use the assembly language of one computer on any other computer.

Q.13. Distinguish between low-level and high-level languages.

High-level languages are easy and low-level languages are difficult. Low-level languages provide more hardware support than high-level languages. The programs written in low-level languages are faster in execution. High level languages provide machine independence.

Q.14. Write the advantages of high level language with respect to machine level language.

High-level languages are easy to learn than machine level languages. The program written in high level languages are shorter than machine level languages. High level languages provide machine independence. It is easier to find errors in the programs written in high level languages. High level languages are closer to human language.

Q.15. List some commonly used high-level languages.

Common high-level languages are C/C++, Java, Pascal, FORTRAN, BASIC, COBOL.

Q.16. How is the program logic implemented in unstructured programming languages?

In unstructured programming languages, the entire logic of the program is implemented in a single module or function. The programs written in these languages are error prone, difficult to understand, modify and debug.

Q.17. How is the program logic implemented in structured programming languages?

In structured programming languages, the entire logic of the program is divided into number of smaller modules or function. Each module is a piece of code that implements a different functionality. There is also a less possibility of errors in the programs.

Q.18. What is meant by language processor? List different types of translators

Language processor or translator is software that converts the programs of high-level languages into machine language. Every computer language has its own translators. Different types of translators include compiler, interpreter and assembler.

Q.19. What is a compiler?

A compiler is a program that converts the instruction of a high level language into machine language as a whole. The compiler converts the source program into machine code. The machine code program is known as object program.

Q.20. How does a compiler work?

The compiler checks each statement in the source program and generates machine instructions. Compiler also checks syntax errors in program. A source program containing an error cannot be compiled.

Q.21. What is an assembler?

An assembler is translating program that translates the instruction of a assembly language into machine language.

Q.22. Differentiate between compiler and interpreter.

The main difference between compiler and interpreter is that compiler converts a program into machine code as a whole and interpreter converts a program into machine code statement by statement.

Q.23. Define source code.

A program written in a high level language is called source code. It is also called source program.

Q.24. Why the source code cannot be executed directly?

The source code cannot be executed by computer directly because the computer does not understand it. It is converted into machine code and then executed.

Q.25. Define object code.

A program in machine language is called object code. It is also called object program. Computer understands object code directly.

Q.26. Distinguish between source code and object code.

Source code is easy to understand and modify. Object code is difficult to understand and modify. Source code contains fewer statements than object code.

Q.27. How the object program, source program and compiler are related?

A source program is written by a programmer that cannot be understood by computer directly. Compiler translates it into object program for computer to understand and execute.

Q.28. What is the use of Turbo C++?

The compiler used for C language is Turbo C++. It is the implementation of Borland International for C language. It is used to create, edit and save programs. It also provides a powerful debugger. The debugger helps users in detecting and removing errors in programs.

Q.29. C is a case sensitive language. What does it mean?

It means that C language can differentiate between uppercase and lowercase words. All keywords are usually written in lowercase.

Q.30. What is the process of linking in C programs?

The process of linking library files with object code is known as linking. The programmer may refer to many files in a C program. The library files must be linked with the object file before execution the program.

Q.31. State the purpose of linker in C language.

The purpose of linker in C language is to combine the object program with additional library files. Linker is part of compiler. It combines object program and library files and saves final machine language program as executable file. The extension of executable file is .exe.

Q.32. List necessary steps taken to prepare a C program for execution.

- Creating and Editing a C Program
- Saving a C Program
- Compiling a C Program
- Linking a C Program
- Loading a C Program

Q.33. Which kind of file is produced when a C program with no syntax error is compiled?

An object file with extension obj is produced when a C program with no syntax error is compiled.

Q.34. Write the shortcut key to compile C program.

The shortcut key to compile C program is ALT+F9.

Q.35. Write the shortcut key to run C program.

The shortcut key to run C program is CTRL+F9.

Q.36. Write the shortcut key to view output screen in Turbo C++ IDE.

The shortcut key to view output screen in Turbo C++ IDE is ALT+F5.

Q.37. Write the name of header file that must be included in program to use the functions printf and scanf?

The header file stdio.h must be included in the program to use the functions print and scanf. This header file contains the definitions of these functions for input and output.

Q.38. What are preprocessor directive?

The preprocessor directives are commands that give instructions to C preprocessor. The preprocessor directives are processed by a program called preprocessor. Preprocessor directives start with hash symbol # and the keyword include or define. These directives are written at the start of program.

Q.39. Give an example of preprocessor directive.

#include <stdio.h>

Q.40. What is the purpose of # sign?

The # sign indicates that it is an instruction for the C preprocessor.

Q.41. What is the purpose of the statement #include <stdio.h> in C program?

The purpose of the statement is to tell the compiler to include the header file **stdio.h** in the program. This header file contains the definitions of built-in input and output functions such as printf() and scanf() etc.

Q.42. What is the purpose of the statement #include <conio.h> in C program?

The purpose of the statement is to tell the compiler to include the header file **conio.h** in the program. This header file contains the definitions of built-in functions such as clrscr() and getch() etc.

Q.43. State the purpose of the statement #include <math.h> in C program.

The purpose of the statement is to tell the compiler to include the header file **math.h** in the program. This header file contains the definitions of built-in mathematical functions such as sqrt() and pow() etc.

Q.44. What will happen if stdio.h file is missing in a C program?

The program will not be compiled if stdio.h header file is missing. It will generate a syntax error when the program uses the standard input and output functions defined in this header file.

Q.45. Name different types of preprocessor directives.

Two types of preprocessor directives used in C language are include and define.

Q.46. What is the use of "include" preprocessor.

The **"include"** preprocessor directive enables a program to access a library. Each library contains different header files. The include preprocessor directive is used to include header files in the program.

Q.47. What is the use of "define" preprocessor.

The define directive is used to define a constant. It starts with the symbol #. It is not terminated with semicolon. It can be used anywhere in the program.

Q.48. What is constant macro? Give an example.

Constant macro is a name that is replaced by a particular constant value before the program is sent to the compiler. #define directive is used to define constant macro as follows:

#define PI 3.142857

Q.49. What are header files?

Header files are collection of standard library functions to perform different tasks. Each header file contains different types of predefined function. The extension of a header file is .h. The include preprocessor directive is used to include header files in programs.

Q.50. Differentiate between preprocessor directives and header file.

The preprocessor directives are commands that give instructions to C preprocessor. They can be used to include files in programs or define constants. The header files are collection of standard library functions to perform different tasks. These files can be included in programs to perform specific functions.

Q.51. What is main function used in C programs?

The main() function is the place where the execution of a C program starts. When the program is executed, the control enters **main()** function and starts executing its statements. Each program must contain **main()** function.

Q.52. What do you know about C statements?

A statement in C language is an instruction for the computer to perform a task. The statements are written in curly brackets. Each statement in C is terminated with semicolon. The compiler generates an error if any statement is not terminated by semi colon.

Q.53. Write a C language statement to print a message "I love Pakistan".

```
printf("I love Pakistan");
```

Q.54. Identify different syntax errors in the following program:

```
include <stdio.h>
void main()
  printf('Hello')
}
```

Ans: 1. # sign missing before include. 2. Starting braces { is missing after main(). 3. The word Hello in third line must be written in double quotes and line must end with semicolon.

Q.55. Define the term bug and debug.

An error in a computer program is known as bug. The programmer can make different errors while writing programs. The errors must be removed from the program before it can be compiled and executed. The process of finding and removing bugs is called debugging.

Q.56. List different types of errors in C programs?

Different types of errors are syntax errors, logical errors and run-time errors.

Q.57. What is a syntax error? Give example.

Syntax error is a type of error that occurs when an invalid statement is written in program. Syntax errors are detected by compiler. A program containing syntax errors cannot be compiled successfully. Typing 'forr' instead of 'for' is an example of syntax error.

Q.58. List different causes of syntax errors.

- The statement terminator is missing at the end of statement.
- A misspelled keyword is used in program
- Any of the delimiters is missing.

Q.59. What are logical errors? Give Example.

A type of error that occurs due to poor logic of the programmer is known as **logical error**. A statement with logical error may produce unexpected and wrong results in the program. Typing a wrong formula to calculate the result is an example of logic error.

Q.60. What are run-time errors?

A type of error that occurs during the execution of program is known as run-time error. It is caused when a statement directs the computer to execute an illegal operation such as dividing a number by zero.

Q.61. Why the logical error is the most difficult error?

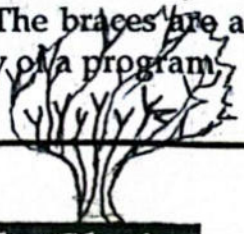
- It cannot be detected by the compiler.
- It does not crash the program. That is why it is difficult to detect.
- The user needs to review the whole program to find out logical error.

Q.62. Differentiate between syntax error and logical error.

A syntax error occurs when an instruction is not written according to the rules of the programming language. A logical error occurs due to poor logic of programmer. The language translator detects syntax errors and display error message to describe the cause of error. Logical errors are difficult to find because translator cannot detect these errors.

Q.63. What do you mean by delimiters?

The statements of the program are written in curly braces. The curly brace { is called opening brace and } is called closing brace. The braces are also known as **delimiters**. These statements are collectively known as the body of a program.



Multiple Choice

1. **A well-defined set of instructions given to the computer is called:**
   a. Hardware b. Software c. Bug d. None
2. **Computer programs are also known as:**
   a. Hardware b. Software c. Firmware d. None
3. **C was developed in the year:**
   a. 1970 b. 1972 c. 1976 d. 1995
4. **B was developed in the year:**
   a. 1969-70 b. 1965-1975 c. 1972-1988 d. None
5. **Who developed C?**
   a. Von-Neumann b. Al-Khuwarizmi
   c. Charles Babbage d. Dennis Ritchie
6. **C was designed to write program for:**
   a. Windows operating system b. Solaris operating system
   c. Unix operating system d. OS/2 operating system
7. **C is a:**
   a. High level language b. Low-level language
   c. Assembly language d. Machine language
8. **Which of the following language provided the basis for the development of C?**
   a. C++ b. B c. JAVA d. Pascal
9. **Turbo C++ can compile:**
   a. C programs only b. C and C++ programs
   c. Turbo C Programs only d. Turbo C++ programs only

Integrated Development Environment

10. An IDE typically consists of:
    a. Text editor b. Compiler c. Debugger d. All
11. An IDE stands for:
    a. Input data error
    b. Input Desktop environment
    c. Integrated Development Environment
    d. Internal disk error
12. The programmer usually enters source code into a computer using:
    a. Compiler b. Text editor c. Debugger d. Linker
13. Which is the correct sequence of steps for creating and executing C program?
    a. Editing → saving → compiling → linking → Executing ESCLE
    b. Compiling → Editing → saving → executing → linking
    c. Editing → Executing → Compiling → Linking
    d. Linking → Executing → saving → Editing → Compiling
14. Which of the following key is used to save a file?
    a. F2 b. F3 c. F5 d. F9
15. The extension of C source program is:
    a. .h b. .c c. obj d. .exe
16. The process of converting source code into object code is knows as:
    a. Compiling b. Executing c. Linking d. Saving
17. The extension of object file is:
    a. .c b. .txt c. .obj d. .h
18. Which file contains the program after translation into machine language?
    a. Object file b. exe file c. Source file d. None
19. Which of the following key is used to compile a program
    a. ALT+F9 b. CTRL+F9 c. CTRL+S d. ALT+F5
20. The output of the compiler is called:
    a. Library code b. Source code
    c. Linked code d. Object code
21. Which of the following key is used to load a program in memory to run?
    a. ALT+F9 b. CTRL+F9 c. CTRL+S d. ALT+F5
22. The process of linking library files with object code is known as:
    a. Compiler b. Executing c. Linking d. Saving
23. The processing of running an executable file is known as:
    a. Debugging b. Compiling c. Executing d. Saving
24. The ".exe" file is produced by:
    a. Linker b. Loader
    c. Compiler d. Interpreter
25. Compiled programs typically execute faster because:
    a. Compiled programs are read and executed a line at a time.
    b. Compiled programs are already in a machine-readable form.
    c. Compiled programs do not require any data.
    d. None of the above
26. The basic structure of C program consists of:
    a. Preprocessor Directive b. main () function
    c. Program body d. All

27. **Processor directives are command for:**
    a. Microprocessor
    b. Language processor
    c. C preprocessor
    d. Loader

28. **The expression in define directive:**
    a. Can only be changed at the end of the program
    b. Cannot be changed
    c. Can not be changed but can be redefined
    d. Cannot be assigned a value

29. **Header files in C contain:**
    a. Compiler commands
    b. Library functions
    c. Header information of C programs
    d. Operators for files

30. **Which of the following is a compiler directive?**
    a. #include <stdio.h>
    b. Using namespace std;
    c. main ( )
    d. All

31. **Which of the following symbol is used to denote a pre-processor statement?**
    a. %
    b. $
    c. #
    d. @

32. **Which of the following syntax is used to include header file?**
    a. #include<name of header file>
    b. # include "name of the header file"
    c. Both a or b
    d. None of these

33. **The expression may be:**
    a. Constant
    b. arithmetic expression
    c. string
    d. All

34. **stdio.h is part of:**
    a. Comment section
    b. C standard library
    c. Compiler
    d. main function

35. **stdio stands for:**
    a. Standard input output
    b. Symbolic input output
    c. Simple input output
    d. String input output

36. **conio stands for:**
    a. Character input output
    b. Console input output
    c. Common input output
    d. Complex input output

37. **The name of header file is written between:**
    a. [ ]
    b. ( )
    c. < >
    d. << >>

38. **The extension of the header file is:**
    a. .CPP
    b. txt
    c. .c
    d. .h

39. **Which header file contains information about standard input / output functions?**
    a. stdio.h
    b. math.h
    c. Both a and b
    d. None

40. **Which of the following header file contains information about common mathematical functions?**
    a. stdio.h
    b. math.h
    c. conio.h
    d. None

41. **This is a complete instruction that causes the computer to perform some action:**
    a. Statement
    b. Line
    c. Keyword
    d. None

42. **For every opening brace in a C program, there must be a:**
    a. Closing brace
    b. Comma
    c. Colon
    d. None

43. Debug is the process of:
a. Creating bugs in program
b. Identifying and removing errors
c. Identifying Errors
d. Removing Errors

44. Which of the following language requires no translator to execute the program?
a. C
b. C++
c. Machine language
d. Assembly language

45. void occupy how many bytes in memory?
a. Zero
b. One
c. Two
d. Four

46. C programs are divided into units called:
a. Functions
b. Section
c. Syntax
d. Debug

47. main is a:
a. Compiler directive
b. Comment
c. Function
d. Expression

48. C statements end with a:
a. Period
b. Comma
c. Semicolon
d. Nothing

49. Which term is commonly used to refer to software or program errors:
a. Crash
b. Short circuit
c. Down
d. Bug

50. Which of the following is NOT an example of a program bug?
a. Run-time error
b. Operator error
c. Syntax error
d. Logic error

51. Mistakes that cause a running program to produce incorrect results are called:
a. Syntax error
b. Logic error
c. Program error
d. None

52. What does the following code print on screen? printf("hello");
a. Hello
b. hello
c. HELLO
d. Nothing

53. A set of rules that must be followed to develop a program is called:
a. Syntax
b. Preprocessor
c. Bug
d. Debug

54. A type of error that occurs when an instruction violates the rules of writing programs is called:
a. Runtime error
b. Logical error
c. Program error
d. Syntax error

55. Which of the following reason causes the syntax error
a. Missing statement terminator
b. using a variable without declaration
c. Missing any of the delimiters
d. All

56. A program's syntax errors is detected by:
a. Compiler
b. Linker
c. Loader
d. Debugger

57. Which of the following errors are NOT detected by compiler?
a. Syntax error
b. Logical error
c. Both a and b
d. None

58. Which of the following does not contain machine language code?
a. Source module
b. Object module
c. Library module
d. None

59. A spelling error (e.g. typing intt instead of int) is an example of:
a. Syntax error
b. Run-time error
c. Logic error
d. None

60. Division by zero (e.g. 2/0) is an example of:
a. Compile error
b. Run-time error
c. Logic error
d. None

61. Machine language:
a. is the language in which programs were first written.
b. is the only language understood by the computer.
c. Differs from one type of computer to another.
d. All of the above

62. A type of language in which instructions are written in binary form is called:
a. Machine language
b. Assembly language
c. High level language
d. None

63. Which of the following is the native language of computer?
a. C language
b. Pascal
c. Machine language
d. DOS

64. Writing programs in machine language is:
a. Tedious
b. Simple
c. Time consuming
d. Both a and c

65. Writing programs in high-level languages is _____ than machine language.
a. Less time consuming
b. More time consuming
c. Less tedious
d. Both a and c

66. Which of the following is NOT a high level language?
a. Assembly language
b. Pascal
c. BASIC
d. FORTRAN

67. The lowest level of programming language is:
a. Java
b. Assembly Language
c. Pascal
d. C++

68. An assembly language uses:
a. English words
b. Mnemonic codes
c. Os and 1s
d. Binary digit

69. Which of the following language is ideal to write business application?
a. COBOL
b. BASIC
c. Assembly language
d. None

70. Which program that translates assembly language code to machine code?
a. Converter
b. Translator
c. Assembler
d. Debugger

71. Which one of the following translates source code to object code as a whole?
a. Interpreter
b. Compiler
c. Debugger
d. Linker

72. Choose the program that translates and executes one line of source code at a time.
a. Compiler
b. Translator
c. Interpreter
d. Assembler

73. The process of identifying and removing errors in the programs is called:
a. Debugging
b. Processing
c. Executing
d. None

74. **Types of translators are:**
a. Compilers b. Interpreters c. Assemblers d. All

75. **A statement that starts with a # is called a:**
a. Preprocessor directive b. keyword
c. Comment d. None

76. **The statements written by the programmer are called:**
a. Source code b. Object code c. Syntax d. None

77. **Computer programs are also known as:**
a. Software b. Procedure
c. Hardware d. Algorithm

78. **Which of the following components is used to convert first.c to first.exe?**
a. Compiler & header b. Compiler & linker
c. Header & linker d. Compiler only

79. **What does the # sign indicate in an instruction of C language?**
a. Instruction for programmer b. Instruction for compiler
c. Instruction for linker d. Instruction for user

Answers

1. b	2. b	3. b	4. a	5. d	6. c	7. a
8. b	9. b	10. d	11. c	12. b	13. a	14. a
15. b	16. a	17. c	18. a	19. a	20. d	21. b
22. c	23. c	24. a	25. b	26. d	27. c	28. c
29. b	30. a	31. c	32. c	33. d	34. b	35. a
36. b	37. c	38. d	39. a	40. b	41. a	42. a
43. b	44. c	45. a	46. a	47. c	48. c	49. d
50. b	51. b	52. b	53. a	54. d	55. d	56. a
57. b	58. a	59. a	60. b	61. d	62. a	63. c
64. d	65. d	66. a	67. b	68. b	69. a	70. c
71. b	72. c	73. a	74. d	75. a	76. a	77. a
78. b	79. b					

Fill in the Blanks

1. A set of instruction given to computer to solve any kind of problem is called Software.
2. A program written in any high level programming language is called _____.
3. In Un structured programming language, the entire logic of the program is implemented in a single module (function).
4. In Structured programming language, the entire logic of the program is divided into number of smaller modules.
5. _____ are commands that give instructions to C preprocessor.
6. The _____ is a program that modifies C program before compilation.
7. A preprocessor directive always begins with the symbol #.
8. The extension of header file is .h.

9. Preprocessor directive is also called compiled directive.
10. Header files are provided by C library.
11. The header file ____ contains information about standard input and output functions.
12. _____ is a name that is replaced by a particular constant value before the program is sent to the compiler.
13. Every statement in a C program must be terminated with a ;.
14. The errors in the program are also known as bug.
15. The process of finding and removing these bugs is called debugging.

Answers

1. computer program	2. source program	3. unstructured
4. Structured	5. Preprocessor directives	6. Preprocessor
7. #	8. .h	9. compiler
10. compiler	11. stdio.h	12. constant macro
13. ;(semicolon)	14. bug	15. debug

True/ False

1. The C programming language was developed by Dennis Ritchie in 1972.
2. C was derived from earlier programming language named B.
3. The B was developed by Ken Thomson in 1980.
4. The shortcut key for compiling a program is Alt+F9.
5. The compiler produces the source file from an object file.
6. The linker is a program that combines the object program with additional files.
7. The shortcut key for executing C programs is Alt+F5.
8. In structured programming, the entire program is divided into smaller modules.
9. High-level language provides machine independence.
10. Modularity is one of the main features of C language.
11. Each statement in C language is terminated by colon or semi-colon.
12. Preprocessor directive is also called compiler directive as it is part of C compiler.
13. The extension of a header file is .hf.
14. The maximum statements that can be written in main() function are 30.
15. A process of converting source code into object code is called compiling.
16. Linking is the process of linking library files with source code.

Answers

1. T	2. T	3. F	4. T
5. F	6. T	7. F	8. T
9. T	10. T	11. T	12. T
13. F	14. F	15. T	16. F