GATE GIST

providing essence....

PROGRAMMING FOR PROBLEM SOLVING

KCS-101/KCS-201

AKTU / UPTU NOTES /

A Complete C-Programming Notes

For B.TECH.-CS/IT Students

BY:- BAKSHI ROHIT PRASAD

(M.Tech. , PhD. – IIIT-Allahabad)

Includes:

- ✓ **Syllabus**
- ✓ **Concepts and Programs**
- ✓ **Important Questions and Answers (For AKTU)**

Programming for Problem Solving

Module – 1 : **(Introduction to Programming)** **[08]**

Introduction to components of a computer system: Memory, processor, I/O Devices, storage, operating system, Concept of assembler, compiler, interpreter, loader and linker.

Idea of Algorithm: Representation of Algorithm, Flowchart, Pseudo code with examples, From algorithms to programs, source code.

Programming Basics: Structure of C program, writing and executing the first C program, Syntax and logical errors in compilation, object and executable code. Components of C language. Standard I/O in C, Fundamental data types, Variables and memory locations, Storage classes.

Module – 2 : **(Arithmetic expressions & Conditional Branching)** **[08]**

Arithmetic expressions and precedence: Operators and expression using numeric and relational operators, mixed operands, type conversion, logical operators, bit operations, assignment operator, operator precedence and associativity.

Conditional Branching: Applying if and switch statements, nesting if and else, use of break and default with switch.

Module – 3 : **(Loops & Functions)** **[08]**

Iteration and loops: use of while, do while and for loops, multiple loop variables, use of break and continue statements.

Functions: Introduction, types of functions, functions with array, passing parameters to functions, call by value, call by reference, recursive functions.

Module – 4 : **(Arrays & Basic Algorithms)** **[08]**

Arrays: Array notation and representation, manipulating array elements, using multi dimensional arrays. Character arrays and strings, Structure, union, enumerated data types, Array of structures, Passing arrays to functions.

Basic Algorithms: Searching &Basic Sorting Algorithms (Bubble, Insertion and Selection), Finding roots of equations, Notion of order of complexity.

Module – 5 :**(Pointer& File Handling)** **[08]**

Pointers: Introduction, declaration, applications, Introduction to dynamic memory allocation (malloc, calloc, realloc, free), Use of pointers in self-referential structures, notion of linked list (no implementation)

UNIT-1

Computer

A computer is an electronic data processing device, which accepts and stores data input in binary format, processes this binary data input, and generates the output in a required format. A computer is a combination of **hardware and software** resources which integrate together and provides various functionalities to the user.

- Hardware represents the physical and tangible components of a computer, i.e. the components that can be seen and touched. Examples of hardware are the following:

  - **Input devices** − keyboard, mouse, etc.
  - **Output devices** − printer, monitor, etc.
  - **Secondary storage devices** − Hard disk, CD, DVD, etc.
  - **Internal components** − CPU, motherboard, RAM, etc.

- Software is the set of programs or instructions that are required by the hardware resources to function properly. Examples of software are the following:

  - **Operating System**
  - **Compiler**
  - **Linker**
  - **Loader, etc.**

Functional Components of a Digital Computer

Working of a computer includes following three major tasks:

- Data Input
- Processing Input
- Data Output

The basic components that helps in performing above mentioned tasks are called as the functional components of a computer. These functional components are:

- Input unit: It takes the input from input devices
- Central processing unit: It does the processing of data
- Output unit: It produces the output and helps in visualizing it.
- Memory unit: It holds the data and instructions during the processing of data.

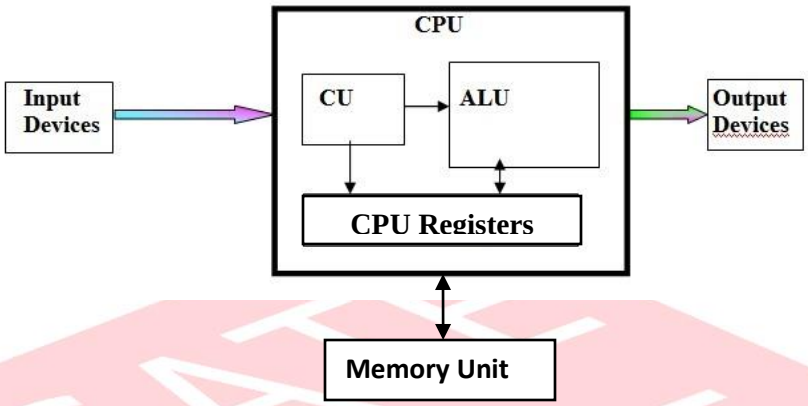


Figure: Block Diagram of a Digital Computer

- **Input Unit:** The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Some of the common input devices are keyboard, mouse, joystick, scanner, etc.

- **Central Processing Unit (CPU):** The CPU is called the brain of the computer because it is the control center of the computer. CPU processes the information input by the input devices. CPU
  - first fetches the instructions from memory and then interprets them so as to know what is to be done.
  - If required, data is fetched from memory or input device.
  - Thereafter CPU performs the required computation
  - After computation, stores the output or displays it on some output device.

Components of CPU

The CPU consists of three main components that are:

- Arithmetic Logic Unit (ALU)
- Control Unit (CU) and
- CPU registers

✓ **Arithmetic and Logic Unit (ALU):** The ALU performs arithmetic and logical operations. Arithmetic calculations include addition, subtraction, multiplication and division. Logical calculation involves relational and logical operations.

✓ **Control Unit:** The CU is responsible for following activities:
  - It coordinates and controls flow of data and instructions among various units of a computer.
  - It controls all the operations of ALU, memory registers and input/output units.

- It takes care of executing all the instructions stored in a program by fetching the instructions, decoding the instructions and sending the appropriate control signals to other functional units until the required operation is completed.

✓ **CPU Registers:** A register can hold the data, instruction, and address of memory location, which is to be directly used by the processor. Registers can be of different sizes (16 bit, 32 bit, 64 bit and so on). There are two major types of registers:
- User/ General Purpose registers: It is used to store operands, intermediate results, etc. during assembly language programming.
- Accumulator (ACC). It is the main register and contains one of the operands of an operation to be performed in the ALU.

- **Output Unit:** It is composed of output devices attached to the computer. It converts the binary output data coming from CPU to human understandable form. Some examples of different output devices are monitor, printer, plotter, etc.

- **Main Memory Unit:** This memory unit stores data and instructions and also called as Primary Memory, Internal memory, and/or Random Access Memory (RAM). Whenever a program is executed, it's data is copied to this memory and is stored in the memory till the end of the execution. This memory is divided into many storage locations also called as memory cells (shown in figure below). Each location is of equal size and can store data or instructions. Also each location has an address associated with it so that computer can read/write any memory location easily.

Address	Memory Locations / Cells
0000	
0001	
.	.
.	.
.	.
.	.
1024	

Storage Devices

Storage device is any hardware capable of holding information either temporarily or permanently There are two types of storage devices used with computers:

- Primary storage device or Primary Memory such as Random Access Memory (RAM) and Read Only Memory (ROM)
- Secondary storage device or Secondary memory such as hard drive, CD, DVD, magnetic tapes, etc.

A. Primary Memory

Memory is the most essential element of a computing system because without it computer can't perform simple tasks. Primary memory is the memory which is directly accessed by the CPU during program execution. The programs and data that the CPU requires during execution of a program are stored in primary memory. Primary memory is of two basic types – Volatile memory and Non-volatile memory.

- **Volatile memory:** The data in a volatile memory is vanished whenever the power supply to it goes off. RAM is an example of volatile memory
- **Non-volatile memory:** The data in a volatile memory is not vanished whenever the power supply to it goes off. ROM is an example of non-volatile memory

1. Random Access Memory (RAM) –

- It is also called as *read write memory*.
- The programs and data that the CPU requires during execution of a program are stored in this memory.
- It is a volatile memory as the data loses when the power is turned off.

2. Read Only Memory (ROM) –

- Stores crucial information essential to operate the system, like the program essential to boot the computer.
- It is a non-volatile memory.
- Always retains its data.

Difference between RAM and ROM

Random Access Memory	Read Only Memory
Temporary Storage	Permanent Storage
Volatile Memory	Non Volatile Memory
Writing data is faster	Writing data is slower
Used in normal operations	Used for startup process of computer

Types of RAM

- **DRAM (Dynamic RAM):** It is made up of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
- **SRAM (Static RAM):** It has a six transistor circuit in each cell and retains its data until power is switched off.
- **NVRAM (Non-Volatile RAM):** It retains its data even when power is turned off. Example: Flash memory.

Types of ROM

- PROM (Programmable ROM)
- EPROM (Erasable PROM)
  EEPROM (Electrically Erasable PROM)

B. Secondary memory

Secondary memory is computer memory that is non-volatile and persistent in nature and is not directly accessed by a computer/processor. Data in secondary memory must be copied into primary storage (also known as RAM) before use. Secondary memories are the slower and cheaper form of memory as compared to primary memory. Secondary memory devices include:

- Magnetic disks like hard drives and floppy disks
- Magnetic tapes
- optical disks such as CDROMs, DVDs, etc.

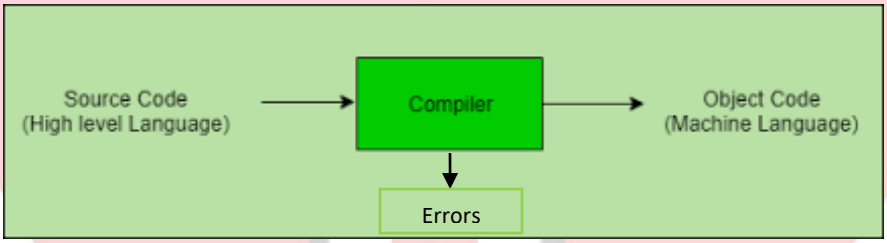
Operating System

- Operating system is a system program that act as an interface between hardware and user.
- It manages system resources.
- It provides a platform on which other application programs are installed.
  Example: Windows, Linux, Unix, DOS, etc.

Functions of Operating System

- Process Management
- Memory Management
- File Management
- I/O Device Management
- Network Management
- Security and Protection

Compiler: Compiler is program that converts the code written in high level languages (like C, C++, etc.) into low level language (like assembly level language or machine level language). Machine language code is the binary code which is understandable by the computer machine. Some examples of compilers for C and C++ languages are Portable C, Turbo C/C++, gcc, Visual C++, Intel C++ (ICC), etc.



Assembler: Assembler is a program that converts the assembly language code into machine language code. For example, Turbo Assembler (TASM), GAS (GNU Assembler), MASM (Microsoft Assembler).

Assembly language: *Assembly language is an extremely basic form of programming. It is a low-level programming language for a computer. Each personal computer has a microprocessor that manages the computer's arithmetical, logical, and control activities. A processor understands only machine language instructions, which are strings of 1's and 0's. However, machine language is too complex for use by humans. So, the low-level assembly language is designed that represents various instructions in symbolic codes which are more understandable by humans. Following are some examples of typical assembly language statements –*

```
INC COUNT        ; Increment the memory variable COUNT

MOV TOTAL, 48    ; Transfer the value 48 in the memory variable TOTAL

ADD AH, BH       ; Add the content of the BH register into the AH register
```

Interpreter: An interpreter is a computer program which translates and executes statements (written in high level language) line by line. It continues translating the program until the first error is met. As it gets first error, it stops. For example, languages such as Python, LISP, Ruby, etc. use interpreters.

Difference Between Compiler and Interpreter

Compiler	Interpreter
It converts high level code into machine code	It converts to machine level code as well as executes it
It has no role in execution	It converts then execute also
Final Output is machine level code	Final output is result
Process entire program: Scans the entire program and translates it as a whole into machine code.	Process Line by line: Translates program one statement at a time
Relatively faster processing	Relatively slower processing
Relatively size is smaller than interpreter	Relatively size is larger
It generates the error message only after scanning the whole program. Hence, debugging is comparatively hard.	Continues translating the program until the first error is met, in which case it stops. Hence, debugging is easy.

Linker and Loader

Linker and Loader are the utility programs that plays a major role in the execution of a program. The Source code of a program passes through compiler, assembler, linker, loader in the respective order, before execution.

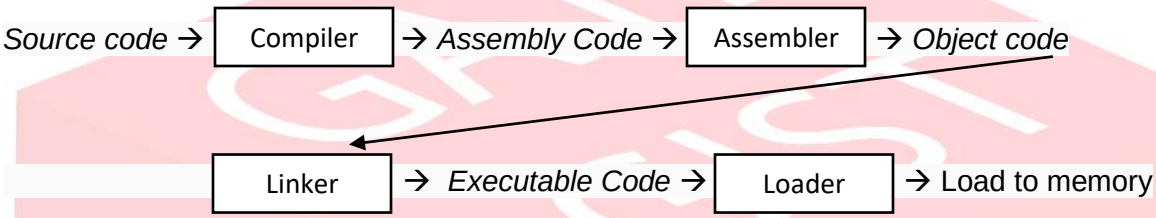


Figure: Execution sequence from source code to loading executable

Linker: The Assembler/Compiler generates the object code of a source program and hands it over to the linker. The linker takes this object code and generates the executable code for the program, and hand it over to the Loader. Linking is performed at the last step in compiling a program. Linker basically performs two major functions:

- **Linking of library functions:** In high-level language programming, the source program may contain some functions whose definition is stored in the built-in libraries or header files. The linker links these functions. In case the built-in libraries are not found it informs to the compiler, and the compiler then generates the error.

- **Linking of object modules:** Sometimes the large programs are divided into the subprograms which are called modules. Now, when these modules are compiled, the object modules are generated. The linker has the responsibility of combining/linking all the object modules to generate a single executable file of the source program.

Loader: As the program that has to be executed must reside in the main memory of the computer, it is the responsibility of the loader to load the executable file of a program (generated by the linker) to the main memory for execution. It allocates the memory space to the executable module in main memory. Following are the main responsibilities of a loader:

- Validate the program for memory requirements, permissions, etc.
- Copy necessary files (such as the program or other libraries) from the disk into the memory
- Link/set the starting point of the program
- Initialize the registers
- Jump to the program starting point in memory

Difference between Linker and Loader

Basis for Comparison	Linker	Loader
Basic	It generates the executable module of a source program.	It loads the executable module to the main memory.
Input	It takes as input, the object code generated by an assembler.	It takes executable module generated by a linker.
Function	It combines all the object modules of a source code to generate an executable module.	It allocates the addresses to an executable module in main memory for execution.

I/O device

An input/output (I/O) device is any hardware used by a human operator or other systems to communicate with a computer. As the name suggests, input/output devices are capable of sending data (input) to a computer and receiving data from a computer (output).

Input device: An input device is any hardware that takes the input (usually from user), converts it into machine understandable binary language and sends it to the processor. In this way, it allows a user to interact with and control the processing. device that sends data to a computer,

Example of Input Devices:

- Keyboard
- Mouse
- Touchpad
- Touchscreen
- Scanner

Output device: An output device is any peripheral that receives the (binary output) data from a processor (CPU), converts it to human understandable form and sends it usually for display, projection, or physical reproduction.

Example of Output Devices:

- Printer
- Headphones
- Monitor
- Printer
- Speaker

Bakshi Rohit Prasad- [M.Tech-IIIT Allahabad]

Algorithm

An algorithm is a sequence of finite and well-defined instructions for completing a task or solving a problem. An algorithm is given an initial state, proceed through a well-defined series of successive states, eventually terminating in an end-state.

Properties of the algorithm

- **Finiteness.** An algorithm must always terminate after a finite number of steps.
- **Definiteness.** Each step of an algorithm must be precisely defined and must be unambiguous
- **Input.** An algorithm has zero or more inputs
- **Output.** An algorithm has one or more outputs
- **Effectiveness.** An algorithm is also generally expected to be effective. This means that all of the operations of algorithm must be so basic that they can be executed in a finite length of time.

Example 1: To display sum of two input numbers:

Algorithm (in Natural Language)

1. Read input numbers num1 and num2

2. Take variable sum

3. Calculate sum = num1+ num2

4. Display value of sum.

5. End

Algorithm (in Pseudocode)

1. Print_Sum (num1, num2)

2. sum ← num1 + num2

3. Print (sum)

4. End

Example 2: To check whether an input number 'n' is even or odd.

Algorithm (in Natural Language)

1. Read input n
2. Calculate the remainder of n/2 and store in variable r
3. If value of r is 0, then number n is even otherwise n is odd.
4. End

Algorithm (in Pseudocode)

1. Check_EvenOdd(n)
2. r ← n%2
3. if r = 0
4. then Print("n is even number")
5. else Print("n is odd number")
6. End

Flow Chart

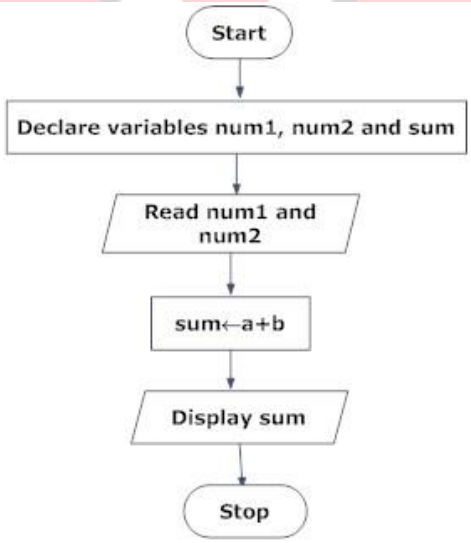
Flowchart is a diagrammatic representation of an algorithm which is the sequence of finite steps for completing a task or solving a problem. Flowchart are very helpful in writing program and explaining program to others.

Symbols used in Flowchart

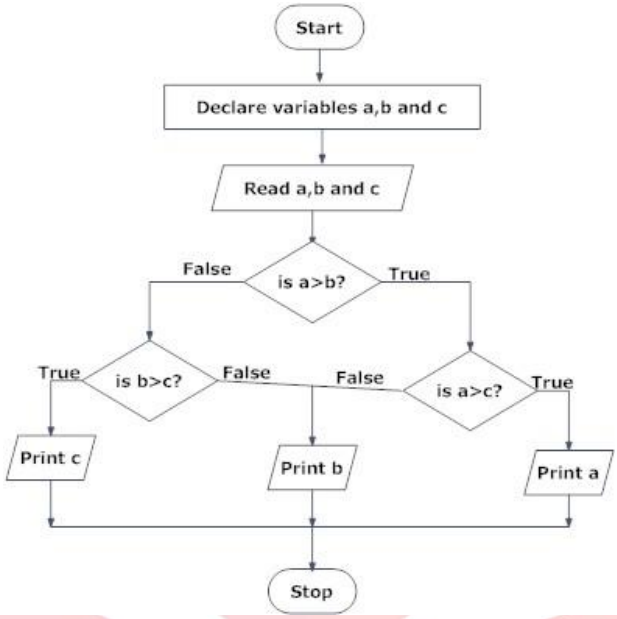
Different symbols are used for different states in flowchart, For example: Input/Output and decision making has different symbols. The table below describes all the symbols that are used in making flowchart

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for airthmetic operations and data-manipulations.
	Desicion	Used to represent the operation in which there are two alternatives, true and false.

Example-1: Draw a flowchart to add two numbers entered by user.



Example-2: Draw flowchart to find the largest among three different numbers.



C-Program Structure

Basic Structure of a C program contains following sections,

Name of Section	**Description of Section**	**Example**
Documentation Section	It consists of set of comment lines giving the name and other details of program	/*Documentation Section: program to find area of circle*/
Link Section	It provides instructions to the compiler to link functions from the system library/header files (system defined and/or user defined). These header files should be included using *#include* directives.	#include<stdio.h> #include<conio.h>
Definition Section	It defines all symbolic constants.	#define PI 3.14
Global Declaration Section	Variables declared in this section are outside of all functions as well as accessible to all functions.	float area;
main() { Declaration Section Executable Section }	Every C program must have one and only one *main()* function section. Execution of the program begins from here. Program contains statements enclosed within the opening brace "{" and closing brace "}". In these two braces main() function contains two parts: **-Declaration Part** declares all the variables used in the executable part. **-Executable Part** contains instructions to perform certain task.	void main() { **/*declaration part*/** float r; **/*executable part starts here*/** printf("Enter radius of circle\n"); scanf("%f",&r); area=PI*r*r; printf("Area of the circle=%f", area); getch(); }
Subprogram section Function1() Function2() …. FunctionN()	It contains all the user defined functions if required.	

Errors: Error is an illegal operation performed by the user which results in abnormal working of the program. Programming errors often remain undetected until the program is compiled or executed. The most common errors can be broadly classified as follows:

1. Syntax errors: Errors that occur when you violate the rules of programming syntax are known as syntax errors. The compiler detects these syntax errors and thus they are known as compile-time errors. Most frequent syntax errors are:

- Missing Parenthesis (})
- Printing the value of variable without declaring it
- Missing semicolon like this

Example Program

```
// C program to illustrate syntax error
#include<stdio.h>
void main()
{
    int x = 10;
    int y = 15;

    printf("%d", (x, y))  // Syntax error: semicolon missed
   getch();
}
```

2. Run-time Errors: Errors which occur during program execution (run-time) even after successful compilation are called run-time errors. These types of errors are hard to find as the compiler doesn't point to the line at which the error occurs. Some of the most common run-time errors are:

- Division by zero
- Null pointer assignment
- Data overflow

Example Program

```
// C program to illustrate run-time error
#include<stdio.h>
void main()
{
    int n = 9, div = 0;

    // Number is divided by 0, so this program abnormally terminates
    div = n/0;

    printf("resut = %d", div);
   getch();
}
```

3. Logical Errors: On compilation and execution of a program, desired output is not obtained when certain input values are given. These types of errors which provide incorrect output despite the program appears to be error free are called logical errors. These errors are easy to detect if we follow the line of execution and determine why the program takes that path of execution.

Example Program

```
// C program to illustrate logical error
void  main()
{
   int i = 0;

   // logical error : a semicolon after loop : This program generates no output.

   for(i = 0; i < 3; i++);
   {
      printf("loop ");
      continue;
   }
   getch();
}
```

Source Code Vs Object Code Vs Executable Code

Source Code: The source code consists of the programming statements that are written by a programmer with the help text editor or a visual programming tool and then saved in a file. For example, a programmer using the C language, types in a desired sequence of C language statements using a text editor and then saves them with a file named MyFile.c. This file is said to contain the source code. It is now ready to be compiled with a C compiler.

Object Code: When the source code file is compiled, it is translated into binary machine language code file which is called as Object Code File/Object File/Object Module. The object code file contains a sequence of instructions that the processor can understand but that is difficult for a human to read or modify. These object files contain unresolved external references (such as printf). They may need to be linked against other object files, third party libraries and almost always against C/C++ runtime library.

Executable Code: The object files/modules are not yet ready for execution. The object file is linked with other object files, third party libraries and runtime libraries. This linking is done by the Linker. After linking, the output of the linker is an executable code which is ready for execution by the processor and may be loaded into memory by the Loader.

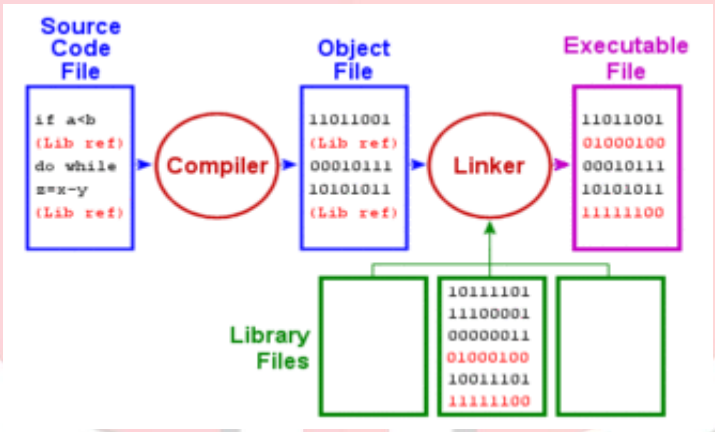


Figure: Source code to Executable code conversion

C Tokens: C Tokens are basic building block of C-programming language. Any C-program is made up of these tokens. C Tokens involves following entities:

- **Identifier:** These are user defined names given to some variable, array, functions, etc. Identifiers follow following properties;
    - Made up of lower-case alphabets (a-z), upper-case alphabets (A-Z), digits (0-9) and underscore _.
    - Must start with an alphabet or underscore
    - Identifiers are case-sensitive i.e. variable named 'Area' and 'area' are not same.

- **Keywords:** These words have fixed meaning and fixed working in the C-program. This meaning cannot be changed. Also, keywords cannot be used as identifiers. For example: for, while, if, else, int, float, char, break, continue, etc. are keywords.

- **Constants:** Constants represent fixed values that may be assigned to some variable Constants are also called literals. Constants can be of any data type. Based on data type constants are of following types:

A. Numeric

a. Integer Constants (e.g., 10, 1, 256, etc.)
   (i) Decimal (ii) Octal (iii) Hexadecimal

b. Real Constants (e.g., 1.34, 0.11, 25.0, etc.)
   (i) floating (ii) double

B. Character

a. **Single Character Constant:** It written in single quotes (e.g., 'a', 'P', '5', '[', '$', etc.)

b. **String Constant**: It written in double quotes (e.g., "Hi!", "Hello friend", "2+1" etc.)

c. Special Character Constant: Also called as escape symbols and have their fixed working. Some examples are as given below:

\n (new line character constant)
\t (horizontal tab character constant)
\v (vertical tab character constant)

- **Strings:** Sequence of characters form a string. Usually strings are written between pair of double quotes “”. For e.g., “Hi!”, “Hello friend”, “2+1” etc. all are examples of strings in C programming.

- **Operators:** Operators are meant to perform some operations on the input operands. There are different types of operators and used for manipulating, calculating, comparing values and for taking logical decisions, etc.

- **Special Symbols:** C supports special symbols that are used for different purposes. Some of the examples and their use is given below:

[ ]	used to declare array
{ }	used to define a block of scope.
( )	used in writing expressions
;	used to terminate or end a statement
=	used to represent assignment operator

Data Types: In the C programming language, data types are declarations for variables that determine the characteristics of the data that may be stored and the methods (operations) of processing that are permitted on them.

Different data types also have different ranges up to which they can store numbers. These ranges may vary from compiler to compiler. The following table provides the details of Fundamental data types with their storage sizes and value ranges (on Turbo C compiler) –

Type	Types of values	Storage size	Value range	Format Specifier in C
char	Stores single character like 'a', 'S', '2', '}', etc.	1 byte	-128 to 127	%c
unsigned char		1 byte	0 to 255	%c
int	Stores integer values like 2, 13, 102, etc.	2 bytes	-32,768 to 32,767	%d
unsigned int		2 bytes	0 to 65,535	%u
short		1 byte	-128 to 127	%hd
unsigned short		1 bytes	0 to 255	%hu
long		4 bytes	-2,147,483,648 to 2,147,483,647	%ld
unsigned long		4 bytes	0 to 4,294,967,295	%lu
float	Stores decimal/real values like 2.426, 0.13, 152.0, etc.	4 bytes	1.2E-38 to 3.4E+38	%f
double		8 bytes	2.3E-308 to 1.7E+308	%lf
long double		10 bytes	3.4E-4932 to 1.1E+4932	%Lf