UNIT I BASICS OF C PROGRAMMING

Introduction to programming paradigms – Applications of C Language - Structure of C program - C programming: Data Types - Constants – Enumeration Constants - Keywords – Operators: Precedence and Associativity - Expressions - Input/Output statements, Assignment statements – Decision making statements - Switch statement - Looping statements – Preprocessor directives - Compilation process

Computer is an electronic device which accepts input, processes data, stores information and produces output.
Data: Raw facts/ figures
Information: Processed data

COMPUTER GENERATIONS

The computer of each generation is smaller, faster and more powerful than preceding generation. There are five computer generations.

First Generation: The vacuum tubes were used for computation. Magnetic drums were used for memory requirements. It consumed lot of space, power. ENIAC (Electronic Numerical Integrator and Computer) used 18000 vacuum tubes, 1800 acquired sq. ft. room space and consumed 180KW of power. The machine level language (0s and 1s) was used. Punched cards were used for input and Paper for output. They were used for scientific work.

Second Generation: The transistors was the most important component which replaced vacuum tubes. Magnetic cores were used for memory. It were more reliable than first generation computer. The assembly or symbolic language was used. The input and output mechanism remained same. The stored program concept was introduced which stores both data and program.

Third Generation: The Integrated circuits(IC) was the most important component. The transistors, diodes, resistors, capacitors were integrated on a single chip. The high-level language was used like BASIC, C, C++ and JAVA. Memory capacity increased and magnetic hard disk was used for secondary generation.
The third generation computers also had OS and computer could run programs invoked by multi users.

Fourth Generation: The Microprocessor was the most important component. With the help of VLSI (Very Large Scale Integration) the entire CPU is on a single chip. OS have moved from MSDOS to GUI (Graphical User Interface) like windows. The networking technology has also been improved. The size was reduced and the speed was increased.

Fifth Generation: Artificial Intelligence and use of natural languages are the main features of this generations. These systems are expected to interact with users in natural language. Speech recognition and speech output should also be possible. Computers must be able to perform parallel processing. The quad-core and octa-core was also introduced. Neural networks and expert systems have been developed.

Classification of Computer Software

Computer software can be broadly classified into two groups.
1. System software
2. Application software

1. System software : It refers to a type of computer software that is responsible for managing and controlling the operation of a computer system. This software is designed to interact with computer hardware at a low level, and it provides the necessary infrastructure and services for other software to run on top of it.

Example:
a) Operating systems: These are the software that manages computer hardware resources and provides common services for other software to run. Examples include Windows, macOS, Linux, and Android.

b) BIOS : Basic Input Output System is a standard defining firmware interface. BIOS built into the computer and is the first code run by the computer when it is switched on.
The key role of BIOS is to load and start the operating system.
When the computer starts, the first function that BIOS performs is to initialize and identify system devices such as Video Display Card, Keyboard, Mouse, Harddisk and DVD Drive.
The code in the BIOS chip runs a series of test called POST (Power On Self Test) to ensure that the system devices are working correctly.

Functions of BIOS:
i) Initializes the System Hardware
ii) Initializes the System Registers
iii) Initializes the Power management system
iv) Tests RAM
v) Tests all Serial and Parallel ports
vi) Initializes CD/DVD Drive and Hard disk controllers
vii) Displays system summary information.

c)Device drivers: These are software that enables the computer to communicate with hardware devices such as printers, scanners, and audio cards.
d) Firmware: This is a type of software that is installed on hardware devices to control their operation. Examples include BIOS (Basic Input/Output System) and firmware on computer peripherals such as keyboards and mice.
e) Utility software: These are software programs designed to perform specific tasks related to system maintenance and management. Examples include disk defragmenters, system cleanup tools, and anti-virus software.

Generation of Computer Languages

1. Machine language / Low-level Language:

A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language, i.e. binary (0s and 1s).
Machine language operates in two different ways:

1. Opcode: An opcode is in charge of giving computers data and information.
2. Location Code: It displays and depicts data and information for storing.

Features of Machine Language
The list below includes a number of the features of machine language.

1. "Low-Level Language" is another name for machine language.
2. In machine language, binary digits such as 0s and 1s are used.
3. There's no need for interpretations since computers can understand binary codes.
4. They were heavily utilized in the very earliest computers.
5. Both learning and using them are complex.

Advantages	Disadvantages
Machine language makes fast and efficient use of the computer.	All operation codes have to be remembered
It requires no translator to translate the code. It is directly understood by the computer.	All memory addresses have to be remembered.
The CPU directly executes the machine language or low-level instructions.	The main drawback of machine language is difficult to develop, learn, and execute codes and algorithms.
The evolution of the computer system & operating system is due to machine language.	It is time-consuming to fix flaws and mistakes in codes and programs.

2. Assembly Language

The operation codes and operands are given in the form of alphanumeric symbols instead of 0's and l's. Assembly language makes use of English like words and symbols. It is also known as 'Symbolic Programming Language.'

These alphanumeric symbols are known as mnemonic codes and can combine in a maximum of five-letter combinations, e.g. ADD for addition, SUB for subtraction, START, LABEL etc.

With the help of special programs called Assembler, assembly language is converted to machine oriented language. Here also a programmer faces practical difficulties because each brand machine had different Opcode.

(ex)

Mov AX, 4	Stores value 4 in AX register of CPU.
Mov BX, 6	Stores value 6 in BX register of CPU.
ADD AX, BX	Adds the contents of AX and BX registers. Stores the result in AX register.

Advantages		Disadvantages
Assembly language is easier to understand and use as compared to machine language.		Like machine language, it is also machine dependent/specific.
It is easy to locate and correct errors.		Since it is machine dependent, the programmer also needs to understand the hardware.
It is easily modified.		

3. High Level Language

High-level computer languages use formats that are similar to English. The purpose of developing high-level languages was to enable people to write programs easily in their own native language environment (English).

Here uses English alphabets, numerals and some special characters. Some of the Higher level languages are FORTRAN, BASIC, COBOL, PASCAL, C, C++, ADA etc.

Each instruction in the high-level language is translated into many machine language instructions by Compiler or Interpreter so that the computer can understand.

Advantages	Disadvantages
High-level languages are user-friendly.	A high-level language has to be translated into the machine language by a translator, which takes up time.
They are similar to English and use English vocabulary and well-known symbols	The object code generated by a translator might be inefficient compared to an equivalent assembly language program.
They are easier to learn.	
They are easier to maintain.	
They are problem-oriented rather than 'machine'-based.	
A program written in a high-level language can be translated into many machine languages and can run on any computer for which there exists an appropriate translator.	
The language is independent of the machine on which it is used i.e. programs developed in a high-level language can be run on any computer text.	

Fourth Generation Languages (4GL)

4GLs are often used in database programming and other applications that require the manipulation of large amounts of data. They are typically non-procedural, meaning that the programmer specifies what results they want rather than how to obtain them. 4GLs also often include features such as built-in query languages and report generators, which can make it easier to develop complex database applications.

Some examples of 4GLs include SQL (Structured Query Language), which is used for managing relational databases, and SAS (Statistical Analysis System), which is used for data analysis and reporting. Other examples include Progress OpenEdge, Oracle's PL/SQL, and Informix 4GL.

Advantages of 4GL:

1. Higher Level of Abstraction: 4GLs provide a higher level of abstraction than traditional programming languages, making it easier for developers to create complex applications with less effort.
2. Increased Productivity: 4GLs have pre-built libraries and tools, which means developers can create applications faster and with fewer errors.
3. Improved Data Management: 4GLs are often used in database programming, which makes it easier to manage large amounts of data.
4. Simplified Coding: 4GLs allow developers to write code that is easier to read and understand than code written in lower-level languages like C or assembly language.
5. Built-in Report Generation: Many 4GLs include built-in report generation tools, which makes it easier to create reports and analyze data.

Disadvantages of 4GL:

1. Limited Control: 4GLs provide a high level of abstraction, which means developers have limited control over the underlying system.
2. Performance Issues: 4GLs are not always optimized for performance, which can result in slower applications or inefficient use of resources.

3. Limited Portability: 4GLs are often platform-specific, which means that applications created with one 4GL may not work on another platform without significant modification.

Fifth Generation Languages (5GL) :

Fifth Generation Language (5GL) is a high-level programming language that was developed during the 1980s to enable non-programmers to communicate with computers in a natural language. 5GLs were envisioned as "intelligent" programming languages that could be used to solve complex problems by allowing users to express their requirements in natural language.

The idea behind 5GL was to provide a more user-friendly programming environment by using a knowledge-based approach that allowed programmers to specify what they wanted the program to do, rather than how they wanted it done. 5GLs were also designed to support distributed computing and artificial intelligence applications.

Advantages of Fifth Generation Language (5GL):

1. Natural Language Interface: The biggest advantage of 5GL is its natural language interface that enables non-programmers to communicate with the computer in their native language. This eliminates the need for learning complex syntax and programming language rules.
2. High-level Abstraction: 5GL provides a high-level abstraction that hides the underlying complexity of the program, making it easier for programmers to focus on the problem at hand.
3. Rapid Application Development: With 5GL, the development of software can be done rapidly and efficiently, allowing programmers to focus on the problem-solving aspect of software development.
4. Supports Artificial Intelligence: 5GL is designed to support artificial intelligence applications, enabling the development of more advanced software that can learn and adapt to new situations.

Disadvantages of Fifth Generation Language (5GL):

1. Limited Scope: 5GLs have a limited scope and are not suitable for developing all types of software applications. They are best suited for solving complex problems in specific domains.
2. Limited Industry Adoption: 5GLs have not been widely adopted in the industry, and most software development is done.
3. Complex Development Process: Developing software in 5GL can be a complex process that requires a lot of knowledge and expertise.
4. High Resource Requirements: Developing software in 5GL may require significant resources, including powerful hardware, software development tools, and specialized expertise.

1.1 Introduction to Programming Paradigms (Models)

Programming paradigms refer to the fundamental styles or approaches of programming that guide the creation of computer programs. Here are some of the most common programming paradigms:

Imperative or Monolithic Programming: This is a programming paradigm where the programmer specifies a series of commands for the computer to perform.

- In this programming paradigm, the whole program is written in a single block.
- We use the goto statement to jump from one statement to another statement.
- It uses all data as global data which leads to data insecurity.
- There are no flow control statements like if, switch, for, and while statements in this paradigm.
- There is no concept of data types.

An example of a Monolithic programming paradigm is Assembly language.

Structure-oriented Programming Paradigm

The Structure-oriented programming paradigm is the advanced paradigm of the monolithic paradigm. It has the following characteristics.

- This paradigm introduces a modular programming concept where a larger program is divided into smaller modules.
- It provides the concept of code reusability.
- It is introduced with the concept of data types.
- It also provides flow control statements that provide more control to the user.
- In this paradigm, all the data is used as global data which leads to data insecurity.

Examples of a structured-oriented programming paradigm is ALGOL, Pascal, PL/I and Ada.

Procedure-oriented Programming Paradigm

The procedure-oriented programming paradigm is the advanced structure-oriented paradigm. It has the following characteristics.

- This paradigm introduces a modular programming concept where a larger program is divided into smaller modules.
- It provides the concept of code reusability.
- It is introduced with the concept of data types.
- It also provides flow control statements that provide more control to the user.
- It follows all the concepts of structure-oriented programming paradigm but the data is defined as global data, and also local data to the individual modules.
- In this paradigm, functions may transform data from one form to another.

Examples of procedure-oriented programming paradigm is C, visual basic, FORTRAN, etc.

Object-oriented programming (OOP): OOP is a programming paradigm that uses objects, which are instances of classes, to represent and manipulate data.

- In this paradigm, the whole program is created on the concept of objects.
- In this paradigm, objects may communicate with each other through function.
- This paradigm mainly focuses on data rather than functionality.
- In this paradigm, programs are divided into what are known as objects.
- It follows the bottom-up flow of execution.
- It introduces concepts like data abstraction, inheritance, and overloading of functions and operators overloading.
- In this paradigm, data is hidden and cannot be accessed by an external function.
- It has the concept of friend functions and virtual functions.
- In this paradigm, everything belongs to objects.

Examples of procedure-oriented programming paradigm is C++, Java, C#, Python, etc.

Other Programming Paradigms:
Functional programming: This is a programming paradigm that treats computation as the evaluation of mathematical functions. It emphasizes immutability and the avoidance of side effects.
Declarative programming: This is a programming paradigm that focuses on describing what the program should do rather than how it should do it.
Logic programming: This is a programming paradigm that uses logical rules to describe relationships and constraints between entities.
Event-driven programming: This is a programming paradigm where the flow of the program is determined by events, such as user input or network communication.
Concurrent programming: This is a programming paradigm that involves writing programs that can execute multiple tasks or processes simultaneously.

Difference Between Procedure Oriented and Object Oriented Programming

Sno	Procedure-oriented	Object-oriented
1	It is often known as POP (procedure-oriented programming).	It is often known as OOP (object-oriented programming).
2	It follows the top-bottom flow of execution.	It follows the bottom-top flow of execution.
3	Larger programs have divided into smaller modules called as functions.	The larger program has divided into objects.
4	The main focus is on solving the problem.	The main focus is on data security.
5	It doesn't support data abstraction.	It supports data abstraction using access specifiers that are public, protected, and private.
6	It doesn't support inheritance.	It supports the inheritance of four types.
7	Overloading is not supported.	It supports the overloading of function and also the operator.
8	There is no concept of friend function and virtual functions.	It has the concept of friend function and virtual functions.
9	Examples - C, FORTRAN	Examples - C++ , Java , VB.net, C#.net, Python, R Programming, etc.

What is C ?
C is a general-purpose computer programming language.
C is also said to be structures programming language or function oriented programming language.
C is a High level programming language.

Why do we use C?

- ❖ C language is used to create applications or software.
- ❖ Initially, C was developed to create an operating system called UNIX.
- ❖ The popular software like **Linux OS**, **PHP** & **MySQL** are created using C language.

Generally C Language is used to create the following…

- Operating Systems
- Language Compilers
- Assemblers
- Interpreters
- Text Editors
- Network Drivers
- Databases

History of C programming language:-
The father of programming languages is ALGOL. It came in the year 1960.
With the help of ALGOL, the developer community learned about the concept of Structured programming language.
In the year of 1967, a new programming language named BCPL was introduced. BCPL stands for Basic Combined Programming Language. It is specially used for writing system software.

After 3 long years, a new programming language called B was created by Ken Thompson. It contained multiple features of BCPL. These two programming languages such as B and BCPL were also known as System programming languages.

In 1972, a great computer scientist named Dennis Ritchie created the C programming language. It includes all the features of **B, BCPL, ALGOL,** and many more additional features too. That's what makes it unique compared to these languages. Most of the UNIX operating systems are created using the C language.
Nowadays, the C is used on a variety of operating systems and hardware platforms. In the year 1990, C was approved by the **International Standards Organization(ISO)**. C language is also known as **'ANSI C'**.

Basics of C Programming

C is a **structured** programming language. So every instruction in a c program must follow the predefined structure (Syntax).
C is also known as **Function Oriented** Programming Language. So every executable statement must written inside a function.

1.1 Applications of C Language

The use of the C programming language is not limited to the development of operating systems and applications. It is also used in GUI development, IDE development etc. Some uses of C language are:-
1. Operating Systems:-
With the help of C, you can write your own operating system. The C programming language is used to develop windows and linux kernels and also the Apple OS X kernel.
2. Graphical User Interface:-
It stands for Graphical User Interface. Apart from creating operating systems, C is also used to develop popular adobe softwares such as photoshop, premier pro, After Effects etc.
3. Embedded Systems:-
In daily life, we use different embedded systems like coffee machines, microwaves, climate control systems etc. C makes it easy to develop these systems.
4. Database:-
Popular database management software, MySQL was developed using the C programming language.
5. Ease of Computation:-
C provides faster computation in programs. The implementation of algorithms and data structures is swift in C. With the help of C, you can perform high degree calculations such as MATLAB, Mathematica etc.
6. Gaming:-
C programming is relatively faster than Java or Python. C language has been used in various gaming applications and graphics. Many popular games like Tic-Tac-Toe, The Snake game, Sudoku etc. are developed using the C language.
7. Development of New languages:-
The C language is fast and easy to understand. So many programming languages like Java, Python, PHP etc. get influenced by this. Even the libraries of Python are developed using C. The syntax and control structures of PERL, PHP and C++ are based upon the C language.
8. Assemblers:-
Assemblers are mainly used to translate assembly level language to machine level language. C also helped in developing GNU assembler.

9. Text Editors:-
C also helped in creating various text editors like Vim, Gedit etc.
10. Interpreters:-
You can also create language interpreters using the C programming langauge. C helped in developing different programming language interpreters like Python and MATLAB interpreters etc.

11. Compiler Design:-
C also helped in designing several popular compilers like Clang C, MINGW, Apple C etc. This is one of the most popular uses of C language.

Features of C:-

1. Portability:- In C, you can execute a block of code in different environments. Suppose, you create a program in one platform and you are running or modifying the program in other platforms. Portability is one of the best features of the C language.
2. Simple and Efficient:- The C programming language is easy to implement. The syntax of the C programming language is easy to understand. If you are confused about which programming language you should learn as a beginner then the C programming language is the best choice for you. It is efficient and simple to learn and use.
3. Speed:- It compiles and executes faster than Java or Python. Because C is a compiler based programming language and Java and Python are an interpreter based programming language.
4. Popular:- The C programming language is used in making operating systems and embedded systems. It is one of the most widely used programming languages in the world.
5. Rich Library:- The C programming language has a rich library which offers useful built-in functions. Apart from these functions, you can also add or define user-defined functions to the library. These functions help in solving numerous types of problems easily and also help in better and clean coding.
6. Dynamism:- The C programming language supports dynamic memory allocation. It is used for utilizing and managing memory. C library has some functions such as malloc(), calloc(), free() etc. to perform dynamic memory allocation.
7. Case Sensitive:- C programming language is case sensitive. If we declare two variables such as "X" and "x" as an integer then the C language treats these two variables differently.
8. Pointer:- Pointer is one of the most useful features in C. With pointers, you can work with the memory in C. You can also use pointers with arrays, structures, functions etc.
9. Recursion:- In C, you can define a function inside of a function. This technique is called Recursion. This will help you in code reusability.
10. Middle-Level Language:- It has features of high-level languages and it has capabilities of assembly language.
11. General-Purpose Language:- The C programming language is being used in various domains like photo editing software to system programming. Also, it is used to develop databases such as MySQL, PostgreSQL, Oracle etc.
12. Structured/Modular Programming language:- In C, you can break the program into small blocks of code with the help of a function. Instead of writing a long and complex code, you can divide the program into small blocks of code as functions then you can perform multiple tasks such as finding the area of square, rectangle, circle etc. A function is used for code reusability.

What is a header file ?
A header file typically has a ".h" extension and is included at the beginning of a source code file using the preprocessor directive "#include". When the code is compiled, the contents of the header file are inserted into the source code file,
Below are some of the examples of header files:-

- stddef.h – Used for defining various types and macros.
- stdint.h – Used for defining exact width integer types.
- stdio.h – Used for defining input and output functions.
- stdlib.h – Used for defining numeric conversion functions, pseudo-random network generator, memory allocation.
- string.h – Used for defining string handling functions.
- math.h – Used for defining mathematical functions.

1.2 Structure of C program

Comments/Documentation Section

- Comments are short explaining the purpose of the program or a statement.
- Generally the documentation part does not executed by compiler & it is optional part.
- The comment begins with /* and ends with */.
- We can put any message we want in the comments.

Example: /* Program to calculate employee salary */

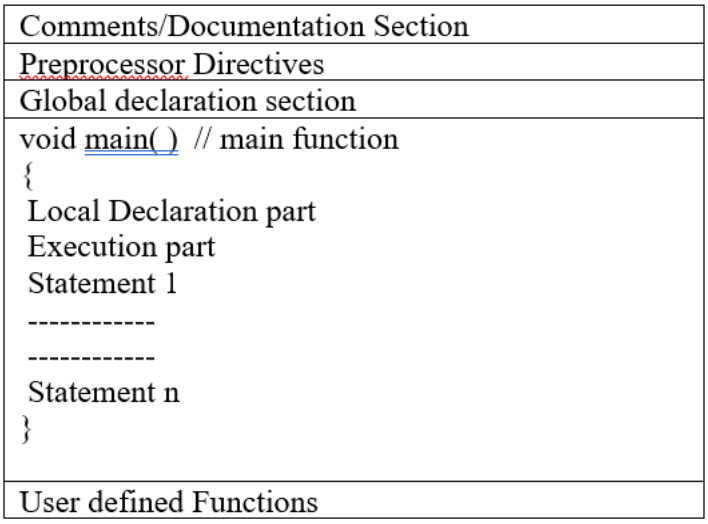
Preprocessor Directives Section

Preprocessor Directives begins with a # symbol. It is used to link the header files, define the constants, etc... Provides instructions to the compiler to include some of the files before compilation starts.

Some examples of header files are :

#include <stdio.h>, #include <conio.h>

#define, #undef, #ifdef, #ifndef, #if, #else, #elif, #endif, #error



Global Declaration Section

This part is used to declare the variables which are common for multiple methods.

In this section, we also declare enumeration, structure, unions, userdefined methods etc...

Main function

Every C program must have only one main function.

The program execution starts with main method and ends with main method itself.

Body of the program

Series of statements that performs specific task will be enclosed within braces { and }

It is present immediately after program header.

It consists of two parts

1. Local Declaration part: This part is used to declare the entire variables that are used in the executable part of the program,

```
ex:   int sum = 0;
      int a , b:
      float b;
```

2. Executable part: It contains at least one valid C Statement.

The Execution of a program begins with opening brace '{' and ends with closing brace '}'

The closing brace of the main function is the logical end of the program.

All statements should end with semicolon(;)

Subroutine Section: Sub programs are basically functions are written by the user (user defined functions).

They may be written before or after a main () function and called within main () function.

This is optional to the programmer.

```
/* Basic C structure example
   To find sum of two numbers */                    //      Documentation Section

#include <stdio.h>                                  // Linker section

int total;                                          //Global variable declaration section
int sum(int, int);                                  // Function declaration

int main( )                                         //main section
{
 int a,b;
 printf("\n Enter 2 numbers");
 scanf("%d %d", &a, &b);
 total=sum(a,b);                                    // calling function. The result is stored in total.
 printf("\nThe sum of two number is %d", total);
 getch();
 }

int sum(int num1, int num2)                         // User defined functions
{
 int result;
 result = num1+num2;
 return(result);
}
```

Mention the Rules or Constraints for Writing C program

- Every c program must contain exact one main method.
- All statements in 'C' program should be written in lower case letters. Uppercase letters are only used for symbolic constants.
- All the statements should end with a semicolon (;)
- Every open brace ({) must have the respective closing brace (})
- The program statements can be written anywhere between the two braces following the declaration part.
- The variables must be declared in the declaration section before they are used.

Compilation and Execution of C program

1. Creating the program
2. Compiling the Program
3. Linking the Program with system library
4. Executing the program

1. Creating the program:

Type the program and edit it in standard 'C' editor and save the program with .c as an extension. This is the source program.

Steps to create Source Code

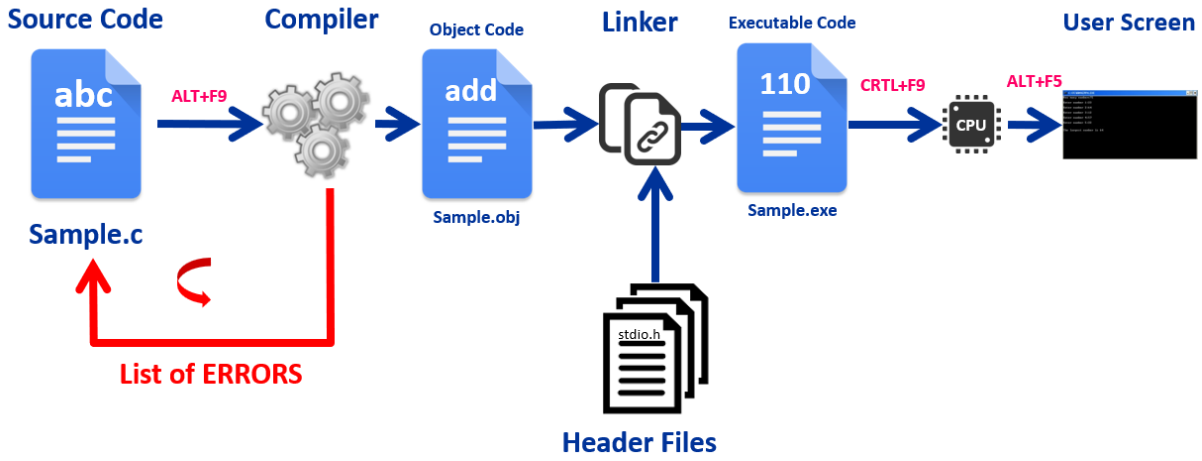
- Open a text editor.
- Create New File.
- Type the program instructions.
- Save the file with ".c" extension.

2. Compiling (Alt + F9) the Program:

This is the process of converting the high level language program to Machine level Language (Equivalent machine instruction).

Errors will be reported if there is any, after the compilation

Otherwise the program will be converted into an object file (.obj file) as a result of the Compilation.
After error correction the program has to be compiled again.



3. Linking the program with system Library:
Before executing a c program, it has to be linked with the included header files and other system libraries -> Done by the Linker

4. Executing the Program:
We need to execute this executable file to generate result .
This process by Running (Ctrl + F9) and testing the program with sample data. If there are any run time errors, then they will be reported.

5. Checking Result
After execution the result is placed in the window called User Screen. We need to open that user screen to check result.
To open user screen in Turbo C, we use shortcut key **ALT + F5**

Variable
A variable is a name given to memory location whose value changes during execution.

Declaration:

Syntax: datatype variable_list;

Where,
datatype may be any built in data type
variable_list - Identifiers that specifies variable name.

Example:
```
int a;
float basic, hra, salary;
```

Variable Initialization:

Syntax: datatype variable_name = value;

Where,
datatype may be any built in data type
variable_name - Identifier that specifies variable name.
value - The data which will be stored in variable name.

Example: int a = 10;

Scope of a variable

A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable cannot be accessed.

There are three places where variables can be declared in C programming language:

1. Inside a function or a block is called local variable,
2. Outside of all functions is called global variable.
3. In the definition of function parameters which is called formal parameters.

Local Variables

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function. Local variables are not known to functions outside their own.

Following is the example using local variables. Here all the variables a, b and c are local to main() function.

```
#include <stdio.h>
int main ()
{
        /* local variable declaration */
        int a, b, c;

        a = 10;
       b = 20;
       c = a + b;
        printf ("value of a = %d, b = %d and c = %d\n", a, b, c);
}
```

Global Variables

Global variables are defined outside of a function, usually on top of the program. The global variables will hold their value throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

Following is the example using global and local variables:

```
#include <stdio.h>
   int g;      /* global variable declaration */

int main ()
{
 /* local variable declaration */
 int a, b;

 a = 10;
 b = 20;
 g = a + b;
 printf ("value of a = %d, b = %d and g = %d\n", a, b, g);
}
```

C Tokens

C tokens, Identifiers and Keywords are the basic elements of a C program. C tokens are the basic buildings blocks in C. Smallest individual units in a C program are the C tokens. C tokens are of six types.

They are,
1. Keywords (eg: int, while),
2. Identifiers (eg: salary, total),
3. Constants (eg: 10, 20),
4. Strings (eg: "total", "hello"),
5. Special symbols (eg: (), {}),
6. Operators (eg: +, /,-,*

1. Keywords

Keywords are also called as reserved words.

- ❖ They already have pre-defined meaning.
- ❖ Keywords should always be written in lowercase.
- ❖ The keywords meaning can't be changed.
- ❖ Keywords cannot be used as identifiers.
- ❖ There are 32 keywords in 'C' program

auto	*double*	*int*	*struct*	*break*	*do*
else	*long*	*switch*	*goto*	*sizeof*	*if*
case	*enum*	*register*	*typedef*	*default*	
char	*extern*	*return*	*union*	*volatile*	
const	*float*	*short*	*unsigned*	*while*	
continue	*for*	*signed*	*void*	*static*	

Identifiers

It refers to the name of the objects.
These are the names given to variables, functions, macros etc.

The rules to write an identifier are as follows:
1. It must contain only alphabets (A to Z, a to z), numbers (0 to 9) and underscore (_).
2. It must start with alphabet or underscore but not numeric character.
3. It should not contain any special symbols apart from underscore.
4. It should not have any space.
5. Keywords cannot be used as identifiers.
6. Identifiers are case sensitive.
7. Maximum length of an identifier is 31 characters.

Examples:

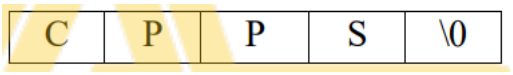
Valid Identifiers: minimum, sum_total, row1, _ tot

Invalid Identifiers:
- float → It is a keyword.
- I am → It has space
- 123_Abc → It is starting with number
- price$ --> It has special symbol.

Strings

A group of characters together form string.

Strings are enclosed within double quotes.
They end with NULL character (\0).



C Character Set

C Character set is a collection of characters supported in C programming language.
C Programming language has a rich set of characters which are used to construct c program instructions.

Alphabets
C Language supports all alphabets of English. It supports both UPPERCASE & lowercase letters
Digits
C Language supports 10 digits to construct numbers. Those 10 digits are 0,1,2,3,4,5,6,7,8,9
Special Symbols
C supports a rich set of special symbols that include symbols to perform mathematical operations, condition checking, white space, back space, etc…

In C Programming Language, the character set follow ASCII (American Standard Code for Information Interchange) code text format.
Every character in C language has the respective ASCII value which is used to convert a character into Binary language.

1.4 C programming: Data Types

Type Conversion

Type conversion is the method of converting one type of data into another data type.
There are two types of type conversion.
1. Automatic type conversion
2. Type casting

Automatic /Implicit Type conversion
This type of conversion is done automatically. The resultant value of an expression depends upon the operand which occupies more space, which means the result value converted into highest data type.
The compiler converts all operands into the data type of the largest operand.
This type of type conversion is done implicitly,this method is called as implicit type conversion.

```
Eg.1
 float a,b,c;
 a=10,b=3;
 c=a/b

output= > c= 3.3
```

```
Eg.2
int a,b,c;
a=10,b=3;
c=a/b;

output= >c=3
```

```
Eg.3
int a;
float b,c;
a=10,b=3;
c=a/b;
output=> c=3.3
```

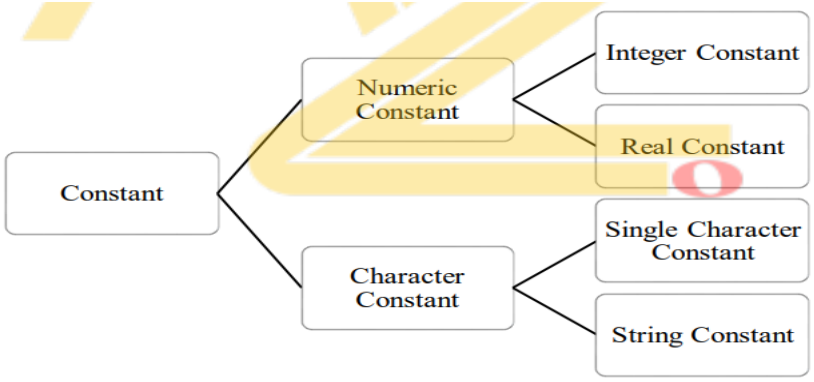
Type casting
This method is used, when user wants to change the type of the data.
General Format for type casting is (datatype)operand.

The type of the x is not changed,only the type of the value can be changed
Since the type of conversion is done explicitly, this type conversion is called as explicit type conversion.

Eg.1
int x=10, y=3;
z=(float)x/y;(ie z=10.0/3;)
output=>z=3.3(float)

Eg:2
int x=10,y=3;
z=x/(float)y;(ie z=10/3.0;)
output=>3.3(float)

1.5 Constants



Numeric Constant : This numeric constant value has not been changed during execution of the program.

Integer Constant : Integer constants are used to represent whole numbers. An integer constant can be specified in decimal, octal, or hexadecimal.
Rules for Constructing Integer Constants
(a) An integer constant must have at least one digit.
(b) It must not have a decimal point.
(c) It can be either positive or negative.
(d) If no sign precedes an integer constant it is assumed to be positive.
(e) No commas or blanks are allowed within an integer constant.
(f) The allowable range for integer constants is -32768 to 32767.

Ex.: 426
+782
-800

1. **Decimal:** It is an integer constant consisting of numbers from 0-9. It can be preceded by + or –
2. **Octal:** It is an integer constant consisting of numbers from 0-7. It is preceded by o

Real constants : They are often called Floating Point constants. The real constants could be written in two forms—Fractional form and Exponential form.

In fractional form:

- A real constant must have at least one digit.
- It must have a decimal point.
- It could be either positive or negative.
- Default sign is positive.
- No commas or blanks are allowed within a real constant.

Ex.: +325.34
426.0
-32.76
-48.579

The exponential form of representation of real constants is usually used if the value of the constant is either too small or too large.
In exponential form of representation, the real constant is represented in two parts. The part appearing before 'e' is called exponent.

- The mantissa part and the exponential part should be separated by a letter e.
- The mantissa part may have a positive or negative sign.
- Default sign of mantissa part is positive.
- The exponent must have at least one digit, which must be a positive or negative integer. Default sign is positive.
- Range of real constants expressed in exponential form is -3.4e38 to 3.4e38.

Character Constants

A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas.

The maximum length of a character constant can be 1 character.
Each character have an integer value called as ASCII (American Standard Code for Information Interchange)
A: 65, a: 97
Ex.: 'A'
'I'
'5'
'='

String Constant:

A group of characters together form string.
Strings are enclosed within double quotes.
They end with NULL character (\0).

Ex: "CBIT" →

C	B	I	T	\0

Qualified Constant/ Memory Constant:

These are created by using "const" qualifier. Const means that something can only be read but not modified.

Syntax: const datatype variable_name = value;
Ex: const int a = 10;
const float pi = 3.14;

Symbolic Constants/ Defined Constant:

These are created with the help of define preprocessor directive.
ex: #define PI 3.142
During processing the PI in the program will be replaced with 3.142.
The name of constant is written in uppercase just to differentiate from the variables.

1.6 Enumeration Constants

In C, an enumeration constant, also known as an enum, is a user-defined data type that represents a set of named values. It allows you to define a list of named constants, called enumerators, which can be assigned unique integer values. Enums provide a way to define symbolic names for integral values, making the code more readable and maintainable.

Here's an example program that demonstrates the usage of enumeration constants:

```
#include <stdio.h>
// Define an enum named Colors
enum Colors {
   RED,    // 0
```

```
    GREEN,  // 1
    BLUE    // 2
};

int main() {
    enum Colors myColor;

    myColor = GREEN;  // Assign the value GREEN to myColor

     if (myColor == RED) {
        printf("The color is red.\n");
    } else if (myColor == GREEN) {
        printf("The color is green.\n");


    } else if (myColor == BLUE) {
        printf("The color is blue.\n");
    }

    return 0;
}
```

In this example, we define an enum named Colors with three enumerators: RED, GREEN, and BLUE. By default, the enumerators are assigned consecutive integer values starting from 0, so RED is assigned 0, GREEN is assigned 1, and BLUE is assigned 2.

Inside the main() function, we declare a variable myColor of type Colors and assign the value GREEN to it.
We then use an if statement to check the value of myColor and print the corresponding color message.

When you run this program, it will output: The color is green.

Enums provide a convenient way to represent a set of related values as named constants, improving code readability and making it easier to understand and maintain.

1.8 Operators

Arithmetic Operator

In the C programming language, arithmetic operators are used to perform various mathematical operations on numerical data types. Here are the arithmetic operators in C:

1. Addition (+): Adds two operands together. Example: **int sum = 3 + 5;** (sum will be 8)
2. Subtraction (-): Subtracts the right operand from the left operand.

Example: **int difference = 10 - 4;** (difference will be 6)

3. Multiplication (*): Multiplies two operands.

Example: **int product = 2 * 6;** (product will be 12)

Operator	Meaning	Example	Result
*	Multiplication	x * y	The product of x and y.
/	Division	x / y	The quotient of x by y.
%	Modulo division	x % y	The remainder of the division x / y.
+	Addition	x + y	The sum of x and y.
-	Subtraction	x - y	The difference of x and y.
+ (unary)	Positive sign	+x	The value of x.
- (unary)	Negative sign	-x	The arithmetic negation of x.

++	Increment	++x x++	x is incremented (x=x+1). The prefixed operator (++x) increments the operand *before* it is evaluated; the postfixed operator (x++) increments the operand *after* it is evaluated.
--	Decrement	--x x--	x is decremented (x=x-1). The prefixed operator (--x) decrements the operand *before* it is evaluated; the postfixed operator (x--) decrements the operand *after* it is evaluated

```
int x, y; x = 5; y = 2;
printf("%d ", x/y); /* will display 2 */
printf("%d ", x%y); /* will display 1, the remainder of the integer division */
```

Increment and Decrement Operators

These are the increment and decrement operators, ++ and – –. The operator ++ adds 1 to its operand, and – –subtracts 1.

The increment and decrement operators can be used only with variables, not with constants. The operation performed is to add one to or subtract one from the operand.

++x; --y; are the equivalent of these statements: x = x + 1;
y = y – 1

When used in prefix mode, the increment and decrement operators modify their operand before it's used. When used in postfix mode, the increment and decrement operators modify their operand after it's used.

x = 10; y = x++; After these statements are executed, x has the value 11, and y has the value 10. The value of x was assigned to y, and then x was incremented.

In contrast, x = 10; y = ++x; these statements result in both y and x having the value 11. x is incremented, and then its value is assigned to y.

Assignment operator

The assignment operator is the equal sign (=). If x = y means , "assign the value of y to x.

Syntax : variable = expression;

When executed, expression is evaluated, and the resulting value is assigned to variable.

Result = a+b;

Compound Assignement

Compound assignment simplifies the coding of a certain type of assignment operations.

Example, x = x+10; can be written as x += 10;

The operator += tells the compiler to assign to x the value of x plus 10.

Operator	Meaning	Example	Result
=	Simple assignment	x = y	Assign the value of y to x
op=	Compound assignment	x += y	x op= y is equivalent to x = x op (y) (where op is a binary arithmetic or binary bitwise operator)

If the two operands have different types, the value of the right operand is converted to the type of the left operand.

An assignment expression has the type and value of the left operand after the assignment.

Multiple Assignment

You can assign the same value to many variables by using multiple assignments in a single statement. For example:

```
a = b = 100;  // equivalent to  a=(b=100);
              // The value 100 is assigned to b and a.
```

Relational Operators and Logical Operators

Every comparison is an expression of type int that yields the value 1 or 0. The value 1 means "true" and 0 means "false." Comparisons use the relational operators listed

Relational Operator

Relational operators are used to compare values and determine the relationship between them. Here are the relational operators.

Operator	Meaning	Example	Result: 1 (true) or 0 (false)
<	less than	x < y	1 if x is less than y
<=	less than or equal to	x <= y	1 if x is less than or equal to y
>	greater than	x > y	1 if x is greater than y
>=	greater than or equal to	x >= y	1 if x is greater than or equal to y
==	equal to	x == y	1 if x is equal to y
!=	not equal to	x != y	1 if x is not equal to y. In all other cases, the expression yields 0.

Logical Operators

These operators are used to perform logical operations on boolean expressions and return a boolean result (true or false). Here are examples of how these operators work:

Logical AND (&&): The logical AND operator (&&) returns true if both operands are true, otherwise it returns false. Here's an example:

Operator	Meaning	Example	Result: 1 (true) or 0 (false)
&&	logical AND	x && y	1 if both x and y are not equal to 0
||	logical OR	x || y	1 if either or both of x and y is not equal to 0
!	logical NOT	!x	1 if x equals 0. In all other cases, the expression yields 0.

```
#include <stdio.h>

int main() {
   int x = 5;
   int y = 10;

   if (x > 0 && y > 0) {
      printf("Both x and y are positive.\n");
   }
   return 0;
}
```

Bitwise Operators

Bitwise operators are used to perform operations on individual bits of integral (integer) types. These operators allow you to manipulate the binary representation of numbers at a bit level.

Operator	Meaning	Example	Result (for each bit position)
&	bitwise AND	x & y	1, if 1 in both x and y
|	bitwise OR	x | y	1, if 1 in either x or y, or both
^	bitwise exclusive OR	x ^ y	1, if 1 in either x or y, but not both
~	bitwise NOT	~x	1, if 0 in x
<<	shift left	x << y	Each bit in x is shifted y positions to the left
>>	shift right	x >> y	Each bit in x is shifted y positions to the right

Ternary Operator

The ? operator works like this: Exp1 is evaluated. If it is true, Exp2 is evaluated and becomes the value of the expression. If Exp1 is false, Exp3 is evaluated.

For example :

```
x = 10;
y = x>9 ? 100 : 200;
```

y is assigned the value 100.

Memory Accessing Operators

The operators are used to access objects in memory. The terms used here, such as pointer, array, structure, etc.,

Operator	Meaning	Example	Result
&	Address of	&x	A constant pointer to x
*	Indirection	*p	The object (or function) pointed to by p
[ ]	Array element	x[i]	*(x+i), the element with index i in the array x
.	Member of a structure or union	s.x	The member named x in the structure or union s
->	Member of a structure or union	p->x	The member named x in the structure or union pointed to by p

A pointer is the memory address of an object. A pointer variable is a variable that is specifically declared to hold a pointer to an object of its specified type.

The first pointer operator is &, a unary operator that returns the memory address of its operand.

```
m = &count;
```

"m receives the address of count."
This address is the computer's internal location of the variable.

Assume that the variable count is at memory location 2000. Also assume that count has a value of 100. Then, after the previous assignment, m will have the value 2000.

The second pointer operator is * unary operator that returns the value of the object located at the address that follows it. For example, if m contains the memory address of the variable count, Now q

```
q = *m;
```

has the value 100 because 100 is stored at location 2000.

Size of Operator: sizeof is a unary compile-time operator that returns the length, in bytes, of the variable. Assume double has 8 bytes.

```
double f;
printf("%d ", sizeof(f));  // will display 8
```

Comma Operator : The comma operator strings together several expressions.

The left side of the comma operator is always evaluated as void. This means that the expression on the right side becomes the value of the total comma-separated expression.

For example, x=(y=3, y+1); first assigns y the value 3 and then assigns x the value 4.

The Dot (.) and Arrow (–>) Operators:

The . (dot) and the –> (arrow) operators access individual elements of structures and unions.

```
struct employee
{
 char name[80];
 int age;
 float wage;
} emp;
```

struct employee *p = &emp; /* address of emp into p */

Example : emp.wage = 123.23

The same assignment using a pointer to emp would be

```
p->wage = 123.23;
```

1.9 Precedence and Associativity

If an arithmetic expression is given, there are some rules to be followed to evaluate the value of it. These rules are called as the priority rules. They are also called as the hierarchy rules.
According to these rules, the expression is evaluated as follows;
Rule 1 :- If an expression contains parentheses, the expression within the parentheses will be performed first. Within the parentheses , the priority is to be followed.
Rule 2 :- If it has more than parentheses , the inner parenthesis is performed first.
Rule 3:- If more than one symbols of same priority , it will be executed from left to right.

C operators in order of precedence (highest to lowest). Their associativity indicates in what order operators of equal precedence in an expression are applied.

Operator	Operation	Associativity	Priority
() [] . -> ++ --	Parentheses Brackets (array subscript) Dot operator Structure operator Postfix increment/decrement	left-to-right	1
++ -- + - ! ~ (type) * & sizeof	Prefix inc/decrement Unary plus/minus Not operator,complement Type cast Pointer operator Address operator Determine size in bytes	right-to-left	2
* / %	Multiplication/division/modulus	left-to-right	3
+ -	Addition/subtraction	left-to-right	4
<< >>	Bitwise shift left Bitwise shift right	left-to-right	5
< <= > >=	Relational less than less than or equal to Relational greater than greater than or equal to	left-to-right	6
== !=	Relational is equal to is not equal to	left-to-right	7
&	Bitwise AND Bitwise exclusive	left-to-right	8
^	Bitwise exclusive OR	left-to-right	9
|	Bitwise inclusive OR	left –to-right	10
&&	Logical AND	left-to-right	11
||	Logical OR	left-to-right	12
?:	Ternary conditiona	right-to-left	13

Example for evaluating an expression

Let X = 2 , Y =5 then the value of the expression
(((Y - 1) / X) * (X + Y)) is calculated as:-
(Y - 1) = (5 - 1) = 4 = T1
(T 1 / X) = (4 / 2) = 2 = T2
(X + Y) = (2 + 5) = 7 = T3
(T2 * T3) = (2 * 7) = 14
The evaluations are made according to the priority rule.

1.10 Expressions

It is combination of operands (variables, constants) and operator.
Evaluation of expressions: Expressions are evaluated using an assignment statement.
Syntax :

variable = expression;

sum = a + b;

Example Program for Arithmetical operator

```
#include <stdio.h>
int main()
{
int m=40,n=20, add,sub,mul,div,mod;
add = m+n;
sub = m-n;
mul = m*n;
div = m/n;
mod = m%n;
printf("Addition of m, n is : %d\n", add);
printf("Subtraction of m, n is : %d\n", sub);
printf("Multiplication of m, n is : %d\n", mul);
printf("Division of m, n is : %d\n", div);
printf("Modulus of m, n is : %d\n", mod);
}
```

Output

```
Addition of m, n is : 60
Subtraction of m, n is : 20
Multiplication of m, n is : 800
Division of m, n is : 2
Modulus of m, n is : 0
```

Example Program for Logical operator

```
#include <stdio.h>
int main()
{
int a=40,b=20,c=30;
if ((a>b )&& (a >c))
{
printf("a is greater than b and c");
}
else
if(b>c)
printf("b is greater than a and c");
else
prinf("c is greater than a and b");
}
```

Output

```
a is greater than b and c.
```

Example Program for conditional operator

```
#include <stdio.h>
int main()
{
int x,a=5,b=3;
x = (a>b) ? a : b ;
printf("x value is %d\n", x);
}
```

Output

```
x value is 5
```

Example Program for sizeof operator

```
#include<stdio.h>
int main()
{
 int ivar = 100;
 char cvar = 'a';
 float fvar = 10.10;
 printf("%d", sizeof(ivar));
 printf("%d", sizeof(cvar));
 printf("%d", sizeof(fvar));
 return 0;
}
```

Output :

```
2    1    4
```

1.11 Input/Output statements

In 'c' language several functions ara available for input/output operations. These functions are collectively known as the standard I/O library.

1.Unformatted input /output statements

2. Formatted input /output statements

Unformatted Input /Output statements

These statements are used to input /output a single /group of characters from/to the input/output devices .Here the user cannot specify the type of data that is going to be input/output.

The following are the Unformatted input /output statements available in 'C'

Input	Output
getchar( )	putchar( )
getc( )	putc( )
gets( )	Puts( )

Single character input - getchar( ) function:

A getchar( ) function reads only one character through the keyboard.

Syntax: char variable=getchar( );

Example: char x;
x=getchar( );

Single character output-putchar( ) function:

A putchar( ) function is used to display one character at a time on the standard output device.

Syntax: putchar(charvariable);

Example: char x;
putchar(x)

getc( ) function
This is used to accept a single character from the standard input to a character variable.
Syntax: character variable=getc( );
Example: char c;
c=getc( );

putc( ) function
This is used to display a single character variable to standard output device.
Syntax: putc(character variable);
Example: char c;
putc(c);

gets( ) function
The gets( ) function is used to read the string from the standard input device.
Syntax: gets(string variable);
Example: gets(s);

puts( ) function
The puts( ) function is used to display the string to the standard output device.
Syntax: puts(string variable);
Example: puts(s)

Proram using gets and puts function

```
#include<stdio.h>
int main()
{
char scientist[40];
puts("Enter name");
gets(scientist);
puts("Print the Name");
puts(scientist);
}
```

output:
Enter Name:Abdul Kalam
Print the Name:Abdul Kalam

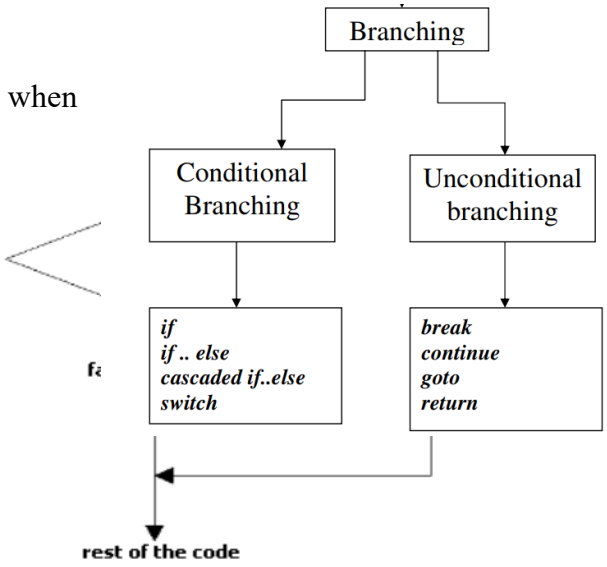
1. Differentiate any two differences between Formatted and Unformatted input statements. (Full differences)

Sno.	Formatted I/O functions	Unformatted functions
1	These functions allow us to take input or display output in the user's desired format.	These functions do not allow to take input or display output in user desired format.
2	These functions support format specifiers.	These functions do not support format specifiers.
3	These are used for storing data more user friendly	These functions are not more user-friendly.
4	Here, we can use all data types.	Here, we can use only character and string data types.
5	printf(), scanf, sprintf() and sscanf() are examples of these functions.	getch(), getche(), gets() and puts(), are some examples of these functions.

1.12 Assignment statements

1.13 Decision making statements

Branching allows your program to execute different blocks of code based on certain conditions.



THE if STATEMENT :

This is basically a "one-way" decision statement. This is used when we have only one alternative.

Syntax

```
if(expression)
{
   statement1;
}
```

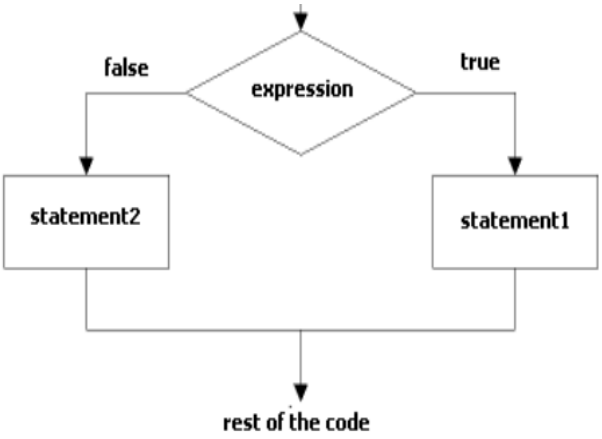
Firstly, the expression is evaluated to true or false. If the expression is evaluated to true, then statement1 is executed. If the expression is evaluated to false, then statement1 is skipped.

THE if else STATEMENT

This is basically a "two-way" decision statement. This is used when we must choose between two alternatives.

Syntax

```
if(expression)
{
   statement1;
}
else
{
  statement2;
}
```



Firstly, the expression is evaluated to true or false.
If the expression is evaluated to true, then statement1 is executed.
If the expression is evaluated to false, then statement2 is executed.

```
#include<stdio.h>
void main()
{
int n;
printf("Enter any non-zero integer: \n") ;
scanf("%d", &n)
if(n>0)
printf("Number is positive number");
else
printf("Number is negative number");
}
```

Output:

```
Enter any non-zero integer:
7
Number is positive number
```

THE nested if STATEMENT

An if-else statement within another if-else statement is called nested if statement.

This is used when an action has to be performed based on many decisions. Hence, it is called as multi-way decision statement.

```
if(expr1)
{
        if(expr2)
                statement1
        else
                statement2
}
else
{
        if(expr3)
                statement3
        else
                statement4
}
```

Program to select and print the largest of the 3 numbers using nested "if-else" statements.

```
#include<stdio.h>
void main()
{
int a,b,c;
printf("Enter Three Values: \n");
scanf("%d %d %d ", &a, &b, &c);
printf("Largest Value is: ") ;
if(a>b)
{
if(a>c)
printf(" %d ", a);
else
printf(" %d ", c);
}
else
{
if(b>c)
printf(" %d", b);
else
printf(" %d", c);
}
}
```

Output:

Enter Three Values:
7 8 6
Largest Value is: 8

switch Statement

This is a multi-branch statement similar to the if – else ladder (with limitations) but clearer and easier to code.

Syntax :

The value of expression is tested for equality against the values of each of the constants specified in the case statements in the order written until a match is found. The statements associated with that case statement are then executed until a break statement or the end of the switch statement is encountered.

When a break statement is encountered execution jumps to the statement immediately following the switch statement.

The default section is optional.

The switch statement however is limited by the following

- Can only test for equality with integer constants in case statements.
- No two case statement constants may be the same.
- Character constants are automatically converted to integer.

Syntax :

```
switch ( expression )
{
    case constant1 :   statement1 ;
              break ;

    case constant2 :   statement2 ;
              break ;

    ...

    default :   statement ;
}
```

```
#include <stdio.h>
void main()
{
 double num1, num2, result ;
 char op ;
 printf ( " Enter number operator number\n" ) ;
 scanf ("%f %c %f", &num1, &op, &num2 ) ;
 switch ( op )
 {
 case '+' : result = num1 + num2 ;
 break ;
 case '-' : result = num1 - num2 ;
 break ;
 case '*' : result = num1 * num2 ;
 break ;
 case '/' : if ( num2 != 0.0 ) {
result = num1 / num2 ;
break ;
}
// else we allow to fall through for error message

 default : printf ("ERROR -- Invalid operation or division
 by 0.0" ) ;
 }
 printf( "%f %c %f = %f\n", num1, op, num2, result) ;
}
```

1.14 Looping statements

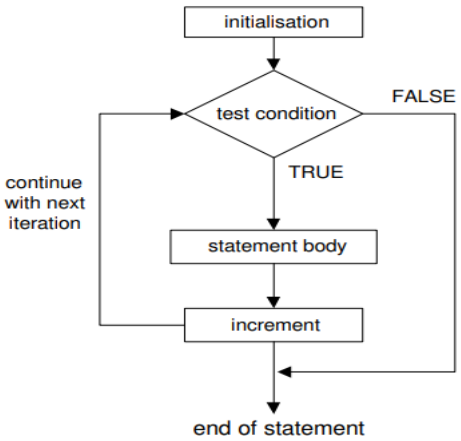
In C programming, a looping statement is used to execute a block of code repeatedly until a certain condition is met. It allows you to automate repetitive tasks and control the flow of your program based on specific conditions. C provides three types of looping statements:

for statement

The for statement is most often used in situations where the programmer knows in advance how many times a particular set of statements are to be repeated.

Syntax : for ([initialisation] ; [condition] ; [increment])
{
 [statement body] ;
}

initialisation :- this is usually an assignment to set a loop counter variable for example.
condition :- determines when loop will terminate.
increment :- defines how the loop control variable will change each time the loop is executed.
statement body :- can be a single statement, no statement or a block of statements.



Example : Program to print out all numbers from 1 to 100.

```
#include <stdio.h>
void main()
{
int x ;
for ( x = 1; x <= 100; x++ )
printf( "%d\n", x ) ;
}
```

While statement

while statement is a control flow statement that allows you to repeatedly execute a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
   // code to be executed
}
```

1. The condition is a Boolean expression that determines whether the loop should continue executing or not. It is evaluated before each iteration of the loop. If the condition is true, the code inside the loop is executed. If the condition is false, the loop is exited, and the program continues with the next statement after the loop.
2. The block of code within the curly braces {} following the while statement is known as the loop body.

3. After executing the code inside the loop body, the program goes back to the beginning of the while statement and checks the condition again. If the condition is still true, the loop body is executed again. This process continues until the condition becomes false.

```
#include <stdio.h>

int main() {
   int count = 0;
   while (count < 5) {
      printf("Count: %d\n", count);
      count++;
   }
   return 0;
}
```

do-while loop:

The do-while loop is similar to the while loop, but it checks the condition after executing the loop body. Therefore, the loop body is executed at least once.

Syntax

```
do {
   // code to be executed repeatedly
} while (condition);
```

```
int i = 0;
do {
   printf("%d ", i);
   i++;
} while (i < 5);
```

1.15 Write a c program to find sum of 10 non negative numbers entered by the user.

```
#include <stdio.h>
int main() {
   int numbers[10];
   int i, sum = 0;

   printf("Enter 10 non-negative numbers:\n");
   for (i = 0; i < 10; i++) {
      printf("Number %d: ", i + 1);
      scanf("%d", &numbers[i]);

      if (numbers[i] < 0) {
         printf("Error: Negative number entered.\n");
         return 0;
      }
      sum += numbers[i];
   }
   printf("The sum of the numbers is: %d\n", sum);
   return 0;
}
```

---

3. Differentiate between break and continue statement in C programming language.

Break Statement	Continue Statement
The Break statement is used to exit from the loop constructs.	The continue statement is not used to exit from the loop constructs.
The break statement is usually used with the switch statement, and it can also use it within the while loop, do-while loop, or the for-loop.	The continue statement is not used with the switch statement, but it can be used within the while loop, do-while loop, or for-loop.
When a break statement is encountered then the control is exited from the loop construct immediately.	When the continue statement is encountered then the control automatically passed from the beginning of the loop statement.
Syntax: break;	**Syntax:** continue;
Break statements uses switch and label statements.	It does not use switch and label statements.
Leftover iterations are not executed after the break statement.	Leftover iterations can be executed even if the continue keyword appears in a loop.

4. In which way switch case statement differs from an if statements.

Switch case: Switch case statements are useful when you have multiple values to compare against a single variable or expression. They provide a more concise and structured way to handle multiple possible values.

If statement: If statements are used when you have a single condition to evaluate, and you want to execute different blocks of code based on whether the condition is true or false. They are more flexible and can handle complex conditions with logical operators.

1.15 Preprocessor directives

In C, preprocessor directives are special commands that are processed by the preprocessor before the compilation of the source code. They begin with a hash symbol (#) and are used to control the behavior of the compiler and perform text manipulations in the source code. Preprocessor directives are evaluated and executed before the actual compilation takes place.

Here are some commonly used preprocessor directives in C:

#include: This directive is used to include header files in your source code. Header files contain function declarations, macro definitions, and other declarations necessary for your program.
Example: #include <stdio.h>

This directive includes the standard input/output library header file (stdio.h), which provides functions like printf() and scanf().

#define: This directive is used to define macros, which are symbolic names representing a value or a code snippet. Macros are replaced by their respective values during the preprocessing phase.
Example: #define MAX_VALUE 100

This directive defines a macro named MAX_VALUE with the value 100. Whenever MAX_VALUE is used in the code, it will be replaced by 100.

#ifdef, #ifndef, #endif: These directives are used for conditional compilation. They allow you to include or exclude blocks of code based on whether a certain macro is defined or not.

Example:

```
#define DEBUG

#ifdef DEBUG
   printf("Debug mode is enabled.\n");
#endif
```

#if, #else, #elif: These directives are used for more complex conditional compilation. They allow you to evaluate expressions and selectively include or exclude code based on the result.
Example:

```
#define VERSION 2

#if VERSION == 1
   printf("Using version 1.\n");
#elif VERSION == 2
   printf("Using version 2.\n");
#else
   printf("Unknown version.\n");
#endif
```

In this example, the #if directive evaluates the expression VERSION == 1. If it's true, it includes the corresponding code. If it's false, it moves to the next #elif directive and so on. If none of the conditions match, the code inside the #else block will be included.

1.16 Compilation process

---

2. Write a C program to find an integer is Palindrome or not.

```
#include<stdio.h>
int main()
{
int n,r,sum=0,temp;
printf("enter the number=");
scanf("%d",&n);
temp=n;
while(n>0)
{
r=n%10;
sum=(sum*10)+r;
n=n/10;
}
if(temp==sum)
printf("palindrome number ");
else
printf("not palindrome");
return 0;
}
```

Output

```
enter the number=151
palindrome  number

enter the number=5621
not palindrome  number
```

---

3. Write a C Program to find the number of vowels , consonants, digits and whitespace.

```
#include<stdio.h>
int main(){
  char string[50];
  int i,vowel=0,digit=0,space=0,consonant=0;
  printf("enter any string includes all types of characters:");
  gets(string);
  for(i=0;string[i]!='\0';i++){
    if(string[i]=='a'||string[i]=='e'||string[i]=='i'||
      string[i]=='o'||string[i]=='u')//checking the char is vowel
      vowel=vowel+1;
    else if(string[i]=='0'||string[i]=='1'||string[i]=='2'||
      string[i]=='3'||string[i]=='4'||string[i]=='5'||
      string[i]=='6'||string[i]=='7'||string[i]=='8'||string[i]=='9')
      digit=digit+1;
    else if(string[i]==' ')
      space=space+1;
    else
      consonant=consonant+1;
  }
printf("vowel=%d digit=%d space=%d consonant=%d", vowel,digit,space,consonant);
  return 0;
}
```
