## PRE REQUISITE TO GET INTO PROGRAMMING

**Bicycle Parts**

What a **user** needs to know!!

What a **Professional** needs to know!!


## BLOCK DIAGRAM OF COMPUTER

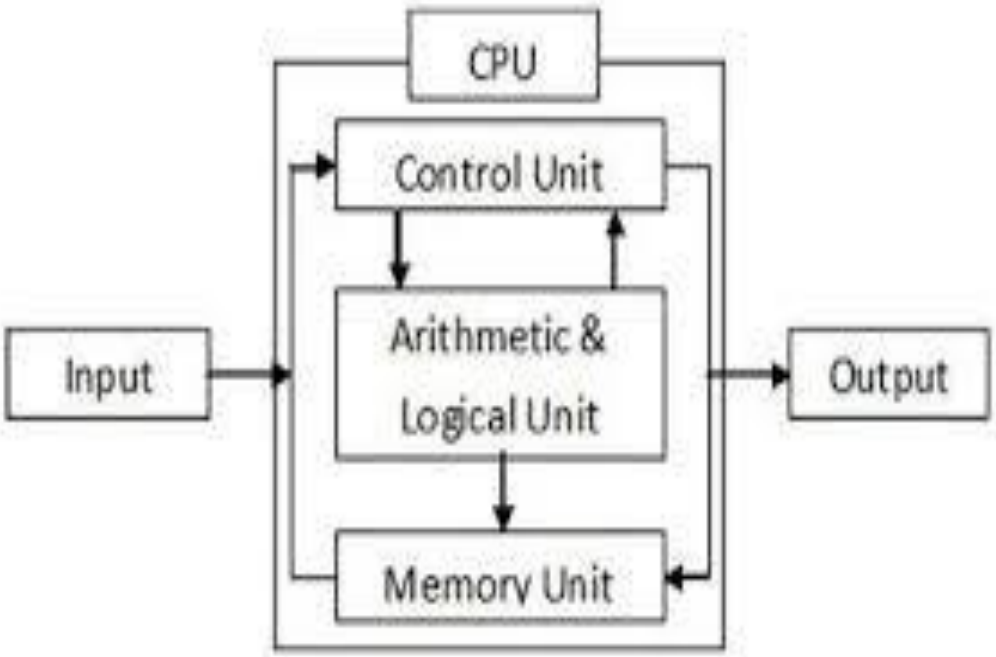
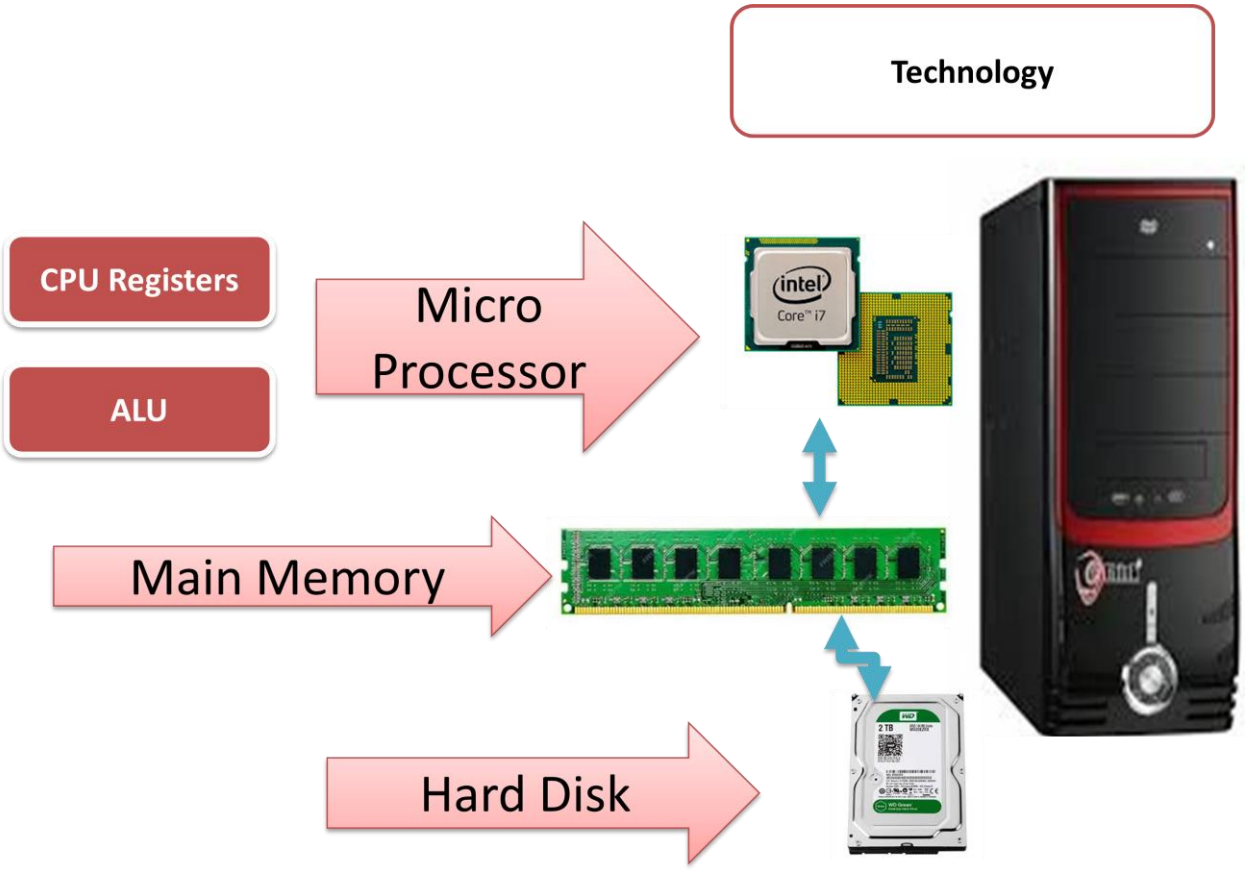
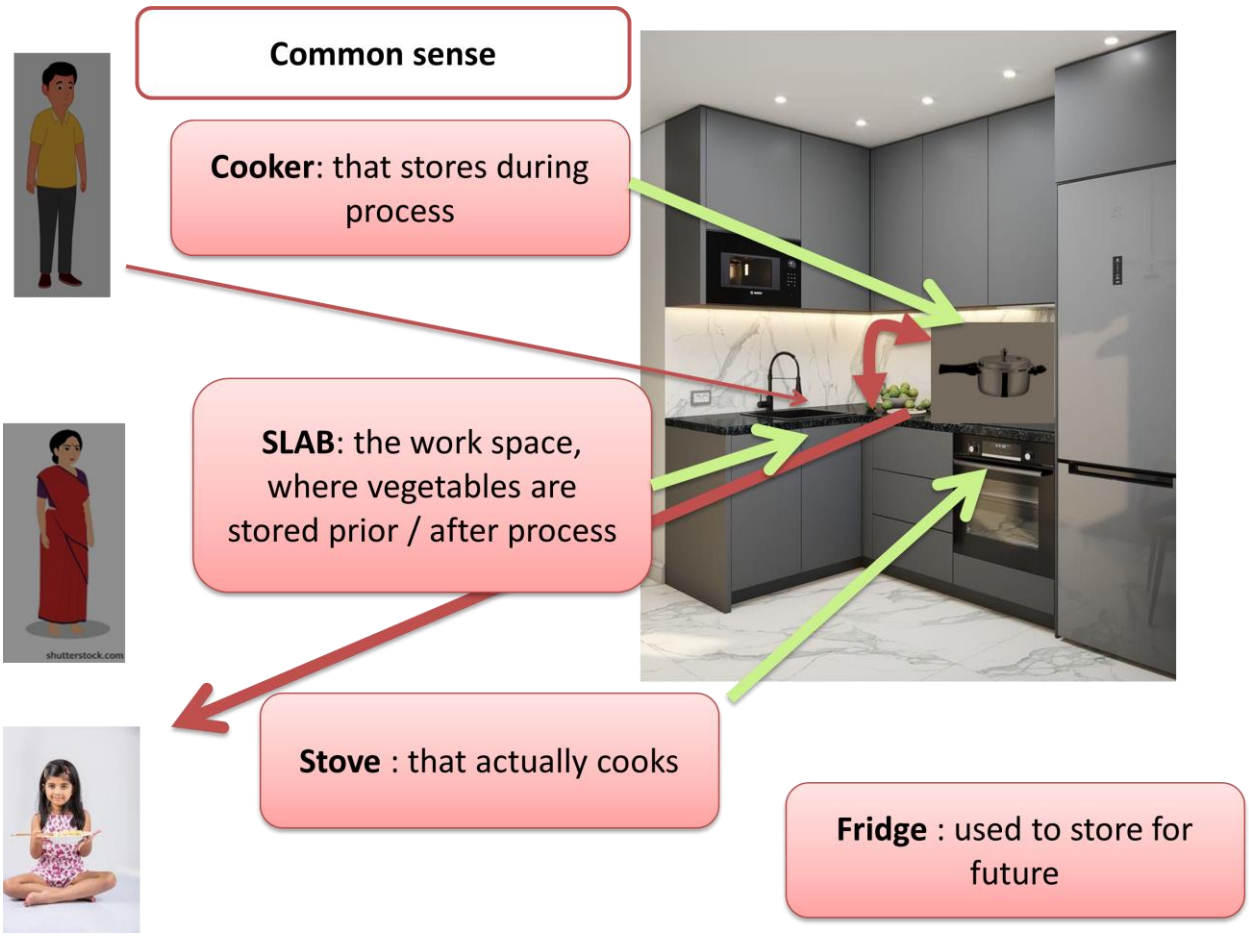



Fig. Block Diagram of Computer


Technology
CPU Registers
ALU
Micro Processor
Main Memory
Hard Disk
intel
Core i7




Machines, like humans, work in **structured stages (because they are made by human )**: **input**, **processing**, **storage**, **control**, and **output**.

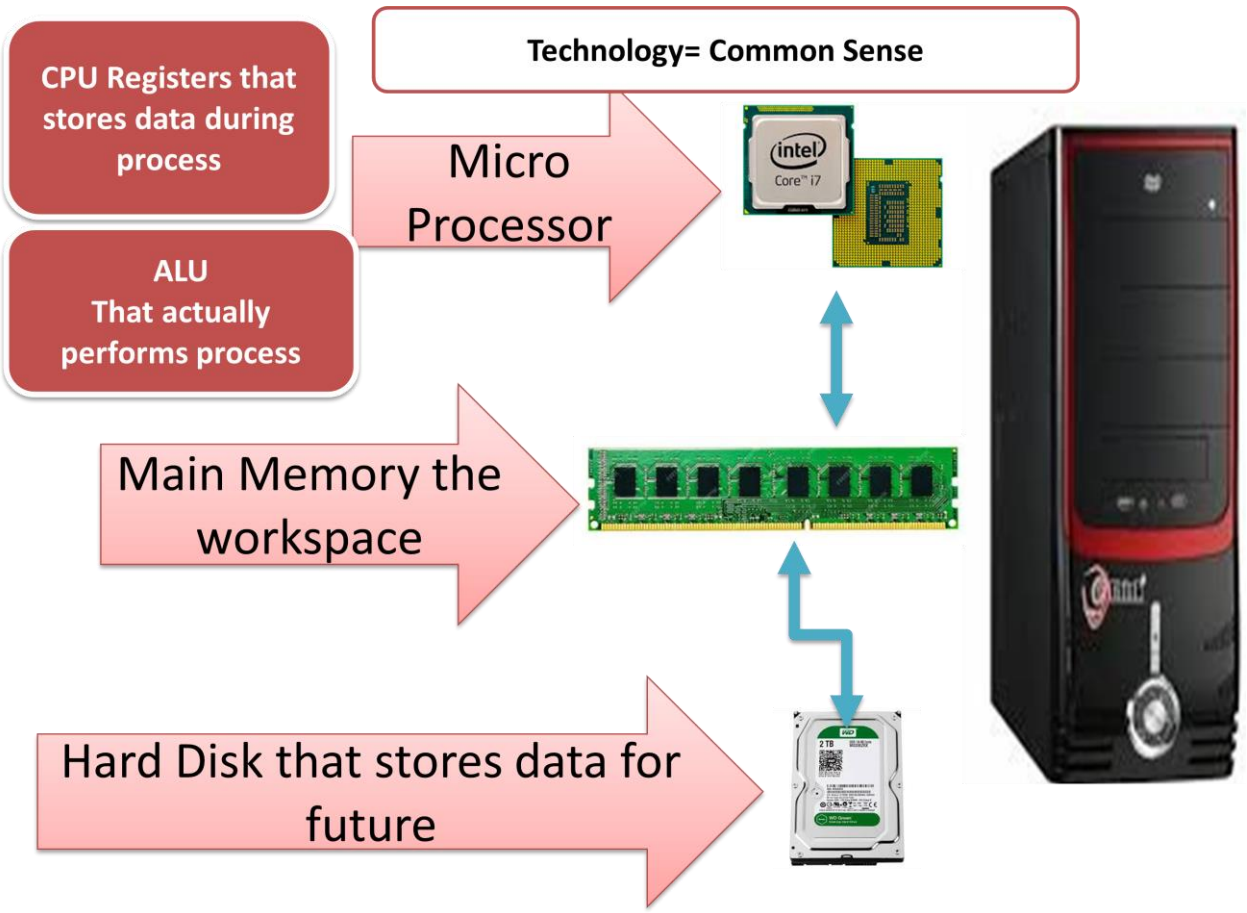
To make this intuitive, let's compare **kitchen setup** with **computer system.**

Each part of a kitchen can represent a functional block of a machine:

- **Stove = ALU (in the Processor - the process done by this)**
- **Cooker = Registers (GPR), stores content / data during process**
- **Fridge = Long-term storage (e.g., Hard Drive)**
- **Slab = Working memory (RAM / workspace)**
- **Mom = Operating System (Coordinates, schedules, decides, organizes)**
- **Dad = Input provider (brings in data/materials)**
- **Daughter = Output (end user / result)**

**Conclusion of the Slide:**

"Just like every person or item in this kitchen plays a defined role to complete a task, a computer system too is made up of well-defined components that work in harmony. Understanding this helps us design more human-aligned AI systems."


Technology= Common Sense
CPU Registers that stores data during process
ALU That actually performs process
Micro Processor
Main Memory the workspace
Hard Disk that stores data for future

## UNIT - 1

Introduction, Compiler and interpreter, Concepts of Machine level, Assembly level and high-level programming, Flowcharts and Algorithms, Fundamentals of C: History of C, Features of C, C Tokens-variables and keywords and identifiers, constants and Data types, Rules for constructing variable names, Operators, Structure of C program, Input /output statements in C, Formatted and Unformatted I/O functions

## INTRODUCTION:

Programming is the process of designing, writing, testing, and maintaining instructions (code) that a computer can execute to perform specific tasks. It involves using programming languages like C, Python, Java, or C++ to create software, applications, and systems that solve problems or automate processes.
Program is defined as set of instructions. The programmers give instructions to system to fulfill the requirement of user, is called programming. The computer languages are used to develop programs, i.e. to give instructions to system.
The computer languages are classified into three types.

i. Machine Level Language (Machine Code) :

This is the lowest-level language. Written only in 0s and 1s (binary). Directly executed on the CPU — no translation needed. Hard for humans to read, write, or debug. Completely hardware-dependent.
Example:

10110000 01100001

ii. Low-Level Language :

Low-level languages are closer to hardware and allow direct manipulation of memory and registers. The languages that are capable of accessing processor features directly are called low level languages. With respect to systems, it is said that the system understands low level languages. There is a problem with low level languages that a program designed in low level language for a specific modal processor cannot be executed on other modal processors because of change in features. This is called hardware dependency. Needs an assembler to convert to machine code.

Ex: Assembly Language

Uses mnemonics like MOV, ADD, SUB.

Example:

```
MOV AX, 05
ADD AX, 02
```

iii. High-Level Language :

To overcome this problem of hardware dependency, high level languages are developed. But high level languages cannot access processor features directly, so they cannot be executed directly in system. With respect to system, it is said that the system doesn't understand high level languages.

For execution the high level programs must be converted into low level code called machine code with the help of compilers or interpreters.

Examples:

C, C++, Java, Python, C#, JavaScript.

Side-by-Side Comparison

| Feature | Machine Level | Low Level | High Level |
|---|---|---|---|
| Representation | 0s and 1s | Mnemonics (MOV, ADD) | English-like syntax |
| Translation Needed | No | Assembler | Compiler/Interpreter |
| Ease of Use | Very difficult | Difficult | Easy |
| Speed | Fastest | Fast | Slower compared to low-level |
| Hardware Dependency | Fully dependent | Dependent | Mostly independent |
| Portability | No | No | Yes |
| Control over Hardware | Maximum | High | Low |

Simple One-Line Differences

Machine Level: Pure binary, directly executed, hardest to program.

Low Level: Assembly language, uses mnemonics, close to hardware.

High Level: Human-friendly language, portable, easy to write.

Compilers and interpreters:

The compilers and interpreters are software mechanisms used to convert high level code into processor appropriate low level code. The compilers convert high level code into low level code line after line and stores as a file on hard disk. The interpreters convert line after line and loads into processor for execution directly. The interpreters won't create a file. When compared compilers are faster than interpreters.

The C language is provided with two compilers. The first compiler converts high level code into intermediate code and saves with extension ".obj" called object file and the second compiler converts intermediate code into low level code, saves with extension ".exe" and further loads into memory for execution. With this two phases of conversion mechanism, the C provides security to high level code.

---

Q. Write about machine , low and high level languages
Q. What are compilers and interpreters and why they are used?

---

**Algorithms:**
The word Algorithm means “a process or set of rules to be followed in calculations or other problem-solving operations”. Therefore Algorithm refers to a set of rules/instructions that step-by-step define how a work is to be executed upon in order to get the expected results.

Key features of algorithm:

i. Clear and Unambiguous: Algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
ii. Well-Defined Inputs: If an algorithm says to take inputs, it should be well-defined inputs.
iii. Well-Defined Outputs: The algorithm must clearly define what output will be yielded and it should be well-defined as well.
iv. Finite-ness: The algorithm must be finite, i.e. it should not end up in an infinite loops or similar.
v. Feasible: The algorithm must be simple, generic and practical, such that it can be executed upon will the available resources. It must not contain some future technology, or anything.
vi. Language Independent: The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be same, as expected.

In order to write an algorithm, following things are needed as a pre-requisite:

o The problem that is to be solved by this algorithm.
o The constraints of the problem that must be considered while solving the problem.
o The input to be taken to solve the problem.
o The output to be expected when the problem is solved.
o The solution to this problem, in the given constraints.

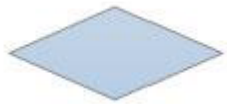
Then the algorithm is written with the help of above parameters such that it solves the problem.
Ex: Algorithm to add 3 numbers and print their sum:

1. START
2. Declare 3 integer variables num1, num2 and num3.
3. Take the three numbers, to be added, as inputs in variables num1, num2, and num3 respectively.
4. Declare an integer variable sum to store the resultant sum of the 3 numbers.

5. Add the 3 numbers and store the result in the variable sum.
6. Print the value of variable sum
7. END

**Flow Charts :**

A flowchart is a visual representation of the steps **in** a program or process using specific symbols. It helps in understanding the logic clearly by showing the flow of control from one step to the next. Symbols like oval (Start/End)**,** rectangle (Process)**,** diamond (Decision)**,** and parallelogram (Input/Output) are used. Flowcharts make debugging easier and improve the clarity of program design. They are widely used in algorithm development and system analysis**.**

| Symbol (Diagram) | Name | Meaning / Function |
|---|---|---|
|  **(Oval)** | **Start / End (Terminator)** | Indicates the start or end of a program/process. |
|  **(Rectangle)** | **Process** | A processing step: calculations, assignments, or operations. |
|  **(Parallelogram)** | **Input / Output** | Used for reading input or displaying output. |
|  **(Diamond)** | **Decision** | Represents a condition resulting in Yes/No or True/False. |
|  **(Arrow)** | **Flow Line** | Shows direction of flow from one step to another. |
| ○ **(Circle)** | **Connector** | Connects different parts of a flowchart (jump to another place). |

| S.NO | Algorithm | Flowchart |
|---|---|---|
| 1. | Algorithm is step by step procedure to solve the problem. | Flowchart is a diagram created by different shapes to show the flow of data. |
| 2. | Algorithm is complex to understand. | Flowchart is easy to understand. |

| | | |
|---|---|---|
| 3. | In algorithm plain text are used. | In flowchart, symbols/shapes are used. |
| 4. | Algorithm is easy to debug. | Flowchart it is hard to debug. |
| 5. | Algorithm is difficult to construct. | Flowchart is simple to construct. |
| 6. | Algorithm does not follow any rules. | Flowchart follows rules to be constructed. |
| 7. | Algorithm is the pseudo code for the program. | Flowchart is just graphical representation of that logic. |

---

Q. Explain algorithm with an example
Q. Explain about flow chart symbols
Q. Explain differences between algorithms and flow charts

---

## Fundamentals of C

**History of C :**

C is a high-level programming language developed under the leadership of Dennis Ritchie in 1972 at AT&T's Bell Laboratories. It was created as part of the development of the UNIX operating system.

Earlier versions of UNIX were written in assembly language, which made portability difficult. To solve this, C was designed as a powerful, flexible, and system-friendly language. Later, about 83% of UNIX was rewritten in C, proving its efficiency and portability.

C evolved from earlier languages B and BCPL, and also introduced many new features designed by Dennis Ritchie himself.

Although C is a high-level language, it also allows direct access to processor-level operations. Because it supports both high-level and low-level programming, C is known as a middle-level language.

**Features of C** :

1. Simple and Efficient : C has a small set of keywords and clean syntax, making it easy to learn and very efficient to use.
2. Structured Programming Language : C supports structured programming with concepts like functions, loops, decision-making, and modular programming.

Programs can be broken down into small functions, increasing clarity and maintainability.
3. Middle-Level Language : C combines features of both high-level and low-level languages. It can work close to hardware (like assembly) and also support higher-level programming (like Java/Python).
4. Portable Language : C programs can be compiled on different computers with very little or no modification. This made C extremely popular for system and application development.
5. Rich Set of Operators : C provides a wide range of operators such as arithmetic, logical, relational, bitwise, and assignment operators.
6. Memory Management Capability : C allows dynamic memory allocation using functions like malloc(), calloc(), realloc(), and free().
7. Fast and Efficient Execution : C programs are close to machine-level code, so they execute very fast. This is why C is widely used in system programming, embedded systems, and operating systems.
8. Extensible and Highly Flexible : You can add your own functions and reuse them anywhere in the program or in other programs.
9. Large Standard Library : C includes many built-in functions for handling strings, files, memory, math, etc.
10. Supports Low-Level Programming : C provides features like pointers, bitwise operations, and direct memory access—useful for system-level tasks.

---

Q. Write about history and features of C language?

---

**Tokens-variables and keywords and identifiers, constants and Data types, Rules for constructing variable names**:

Though all the following are not considered as tokens, but these are other building blocks that are mandatorily understood to write programs. As per the basic definition of the term token -"pre defined tools", we can consider the following also as part of tokens.

**Variables :**
During program execution, the programmer needs to reserve some memory in RAM, to store data into that, to do process on it. As this reserved memory is capable of varying its value it is called a variable. The variables are declared for data types, and identified with an identifier (name).
Ex : int x; here 2 bytes memory is reserved of int type and it is named "x".

**Key Words:**
Every programming language has its own set of pre-defined important words called keywords, used to give instructions to system. Each keyword is reserved for a

specific purpose. The keywords can be used only for that purpose for which they are reserved. The C language has 32 keywords. These cannot be used as identifiers.

**Identifiers :**
Identifier is a name given to an entity in program. Generally the variables, functions, structures etc are identified with names and these names are called identifiers.

There are some rules for identifiers:

o In C, identifier must have less than 8 characters (on unix it is < 8, but in windows it can be up to 32 characters)

o Identifier must start with alphabet and can have numbers in between.

o No special symbol including space is permitted other than underscore

o Keywords are not permitted

o Duplicate names not permitted

**Punctuations:**
Punctuation in C programming refers to special symbols used to structure code and define its syntax. These symbols help separate statements, group expressions, and indicate various programming constructs.
1. Semicolon (“ ; “) – Used to terminate statements.
2. Comma (“ , “) - Used to separate multiple values or parameters.
3. Period (“ . “) - Used to access members.
4. Colon (" : ") - Used in labels
5. Parentheses ("( )") - Used in function definitions, function calls, and grouping expressions.
6. Curly Braces ("{ }") -Used to define the scope of functions, loops, and conditions.
7. Square Brackets ("[]") - Used for array indexing.
8. Angle Brackets ("< >") - Used in header file inclusion.
9. Asterisk ("*") -Used for pointer declarations and dereferencing.
10. Ampersand ("&") - Used to get the address of a variable and referencing.
11. qualifier ("->") - Used to access members via pointers.
12. Hash ("#") - Used for preprocessor directives.
Constants :
Fixed values that do not change during execution.
Ex: Integer constants (10, -5), Floating-point constants (3.14, -0.001), Character constants ('A'), String constants ("Hello").
Operators: are symbols used to perform operations on variables and values like +,-,etc

**Basic Data types :**

The data types are used to specify how the data is to be treated. These are basically used while declaring variables.
char : this is used to store text data. The variable of char data type occupies 1 byte memory in RAM and can store between -128 to 127 (ASCII value specification), i.e. one single character.
int : used to declare variables of integer type. It stores whole numbers and occupies 2 bytes memory (on 64 bit systems it occupies 4 bytes). Its range is between -32768 and 32767 (on 32 bit OS).
float : used to store real numbers. It occupies 4 byte memory, and supports 6 precisions only. This is used for general calculations.
double: used to store real numbers and occupied 8 bytes memory. It supports 10 precisions in the value. This is used for scientific calculations.

Format Specifiers
%c : single char
%s : string (group of characters)
%d : decimal format
%i : integer format
%u : unsigned format
%o : octal formal
%x : hexa-decimal format
%f : floating value
%e : scientific notation

**Functions & libraries :**
A function in simple terms is work or process. In C a function is identified with a pair of parenthesis. A function typically has 2 parts, a definition and a call. Function definition includes the process to be executed and the function call is invocation or execution of the definition. C has many pre-defined functions and we can make a call of them and use them. These functions are stored on hard disc in the form of files called library files and identified with extension .h, like stdio.h, conio.h, etc. these library files must be linked to our programs to make use of their functions in a location called pre-processor, which is identified with # to the RE.

---

Q. Explain Tokens of C

---

**OPERATORS IN C**

The operators are used to develop process. Operators are basically classified into three types based on number of operands they take.

1. Unary operators
2. Binary Operators

3. Ternary Operator.

Unary Operators :
the operators that take single operand are called unary operators.
++ (increment) , -- (decrement) , - (negation operator).

Binary Operators :
The operators that take two operands are called binary operators. These are further classified into five types based on their purpose, viz.

**Arithmetic Operators** : these are used to perform basic arithmetic operations on numbers.
+, - , *, / and % (Modulus)
NOTE: The modulus operator can be used only with integer data type.

**Comparison operators**: these are also called relational operators. These operators take two operands and give a result of either true or false and called Boolean values
<, <=, >, >=, == , !=.
These expression developed using these are called either a condition or a Boolean expression.

**Logical Operators**: these operators are used to combine two conditions.
&& (Logical and), || (logical or) and ! (logical not).

**Assignment Operator**: this is simple =. This operator assigns R.H.S value to L.H.S variable. So L.H.S. must be a variable.
For example, int a =5; is a correct statement. But int a; 5=a; is wrong.
The arithmetic operators along with assignment operator perform dual operations.
For example a=a+2 can be written as a+=2.
The operators +=, -=, *=, /=, %= can be used.

Ternary operator: the operator is ? :
Syntax : (condition) ? statement1 : statement2;
The condition is developed using comparison operators, and they give a result of either true or false. Based on the result, if the result is true, the statement 1 will be executed by skipping the statement2, and vice versa.
Ex : for displaying user about biggest of two numbers, the code would look like,
(a>b)?printf("a is big"):printf("b is big");

---

Q. Write about operators in C

---

**Structure of 'C' Program:**

The C program general structure looks as :

Comment section
Preprocessor
Global data members (variables) (if required)
Functions including statements and expressions

The comment section can be placed wherever required. These are used to make sure programmer understands what he does in program when revisiting. In C++ editor, the comments can be either single lined or multi lined. The single line comments are placed with // and multi lined comments are placed within /* */

**Basic IO statements in C:**

The User Interface is providing interactivity between user and system. It is managed with IO statements (instructions). The libraries stdio.h and conio.h have functions pre-defined to handle IO operations.

printf() : is a standard output function, used to produce output onto monitor.
Syntax:

printf("message to be displayed");
printf("message & format specifiers", source variables / values);

ex:

printf("Welcome to C");

scanf() : is standard input function, used to read data from keyboard.
Syntax:

scanf("format specifiers", address of variables);

The main memory (RAM) includes no. of bytes memory , where each byte is identified with a byte number. When a variable is declared, the first byte number of reserved memory is called address of that variable and it is retrieved with the operator & in C.
Ex:

```
int a;
scanf("%d",&a);
```

---

Q. Explain structure of C program with basic io statements.

---

**Formatted and Unformatted I/O functions :**

**IO buffers**: In System, each hardware component has its own personal memory called buffers, in which data is stored either before getting into RAM or after coming out of RAM. For example, the Key Board has its own buffer, in which data is stored when user typed and gets into RAM when generally user hits ENTER. The content

when displayed onto monitor, it is placed ontop monitor buffer, and that exists on monitor buffer are displayed onto monitor.

**Standard and Console IO functions** : the IO functions in which the buffers are managed in a standard procedure are called standard IO functions. i.e. the user needs to type data and hit ENTER.
The printf(), scanf() etc functions follow this standard procedure, so, called standard IO functions and defined in the library stdio.h.

The IO functions that do not follow standard procedure work in a different way, like the getche() bypasses the key board buffer which reading data. The getche() function directly reads data from the console bypassing the buffer, so user need not hit enter. This function is uised to read single char input from console input device. But the char typed by user is reflected onto monitor, as this function doesn't bypass monitor buffer. The function getch(), bypasses both keyboard and monitor buffers, so user need not hit ENTER after data and also the data is not visible on monitor. These kind of functions are called console IO functions and defined in conio.h library.

**Formatted and Unformatted IO functions** : the IO function in which user can use format specifier are called formatted IO functions. Ex: printf(), scanf() etc. but the IO functions like getchar() etc do not take format specifiers. These are format specific. That is, they work only with specific formats.

getchar(): used to read single char
putchar() : used to display single char
gets() : reads a string
puts() : displays a string

---

Q. Explain different types of IO Functions
Q. Write about formatted and unformatted IO functions

---