Unit ⇒ 2

DATE
PAGE

Algorithms :- Algorithm are step by step process or procedures designed to solve specific problems and perform tasks efficiently in the realm of computer science and mathematics.

An algorithm is a set of instructions that are followed to perform a task or solve a problem. Algorithm are used in many areas, including computer programming, mathematics, and artfical intelligence (AI).

Here are some eacamples of algorithm :-

① Search engines use algorithms to rank website for keywords.

② Merge sort algorithms divide and merge element to provide faster results.

Types of Sorting Algorithm :-

① Bubble sort Algorithm
② Selection sort Algorithm
③ Insertion Sort Algorithm
④ Merge Sort Algorithm
⑤ Quick Sort Algorithm
⑥ Heap Sort Algorithm

① Bubble Sort Algorithm :- It is a simple sorting technique to arrange a list of elements in a specific order, usually ascending or descending.

② Selection Sort Algorithm :- It is a straightforward, in place comparison - based sorting technique. The idea behind it is to divide the list into two parts : The sorted part at the start of the list and the unsorted part at the rest of the list.

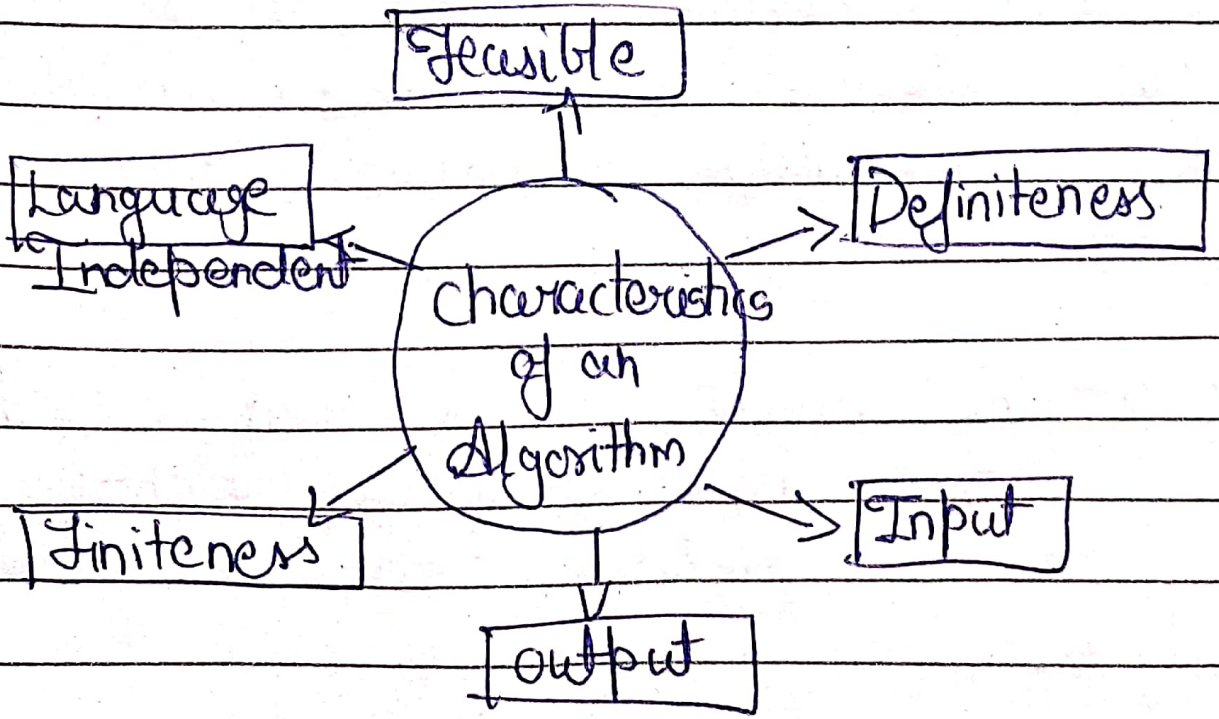
③ Insertion Sort Algorithm :- It is a simple and intuitive sorting algorithm that builds the final sorted list one item at a time.

④ Merge Sort Algorithm :- It is a highly efficient, comparison - based, divide - and - conquer sorting algorithm.

⑤ Quick Sort Algorithm :- It is a highly efficient sorting algorithm based on the divide - and - conquer principle. It works by selecting a 'pivot' element from the array and ~~partion~~ partitioning the other element into two sub- arrays, according to whether they are less than or greater than the pivot.

DATE
PAGE

⑥ Heap Sort Algorithm :- It is a comparison - based sorting algorithm that uses a binary heap data structure.



Characteristics of an Algorithm :-

1. clear and Unambiguous
2. Well defined Inputs
3. Well defined Outputs
4. Finite-ness
5. Feasible
6. Input
7. Output
8. Definiteness
9. Finiteness
10. Effectiveness

Advantages of Algorithms :-

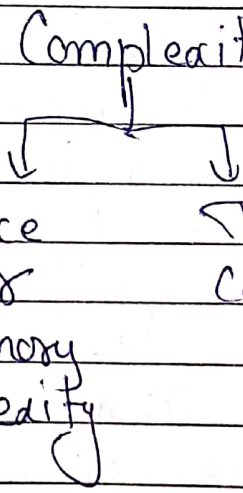
1. It is easy to understand
2. An Algorithm is a step wise representation of a solution of a given problem.

Disadvantages of Algorithms :-

1. Writing an algorithm takes a long time so it is time consuming.
2. Understanding complex logic through algorithms can be very difficult.

Algorithm Complexity :-

Complexity in Algorithms refers to the amount of resources (such as time or memory) required to solve a problem or perform a task. The most common measure of complexity is time complexity, which refers to the amount of time an algorithm takes to produce a result as a function of the size of the input. Memory complexity refers to the amount of memory used by an algorithm



Case in Complexity :-

(1) Worst ~~time~~ Case Complexity :- The function defined by the maximum number of steps taken on any instance of size n.

(2) Best Case Complexity :- The function defined by the minimum number of steps taken on any instance of size n.

③ Average Case Complexity :- The function defined by the average number of steps taken on any instance of size n.

* Time Complexity :-

① Time taken by the algorithm to solve the problem. It is measured by calculating the iteration of loops, number of comparisons etc.

② Time Complexity is a function describing the amount of time an Algorithm takes in terms of the amount of input to the algorithm.

* Space Complexity :-

① Space complexity is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the Algorithm.

② Space Complexity is sometimes ignored because the space used is minimal and / or obvious, but sometimes it becomes as issue as time.

Concepts of OOPs :-

OOP stand for object oriented programming. Individual object are grouped into classes. OOPs implement real world entities like inheritance, polymorphism, hiding etc. into programming. It also allow binding data and code together.

The main features of OOPs Or Concept of OOPs :-

1. class
2. object
3. Inheritance
4. Encapsulation
5. polymorphism
6. Abstraction
7. Message passing
8. Dynamic binding.

① class :- A class is a blueprint or template of an object. It is logical entity. It is the best example of data binding. It is a user defined data types inside a class, we define data type variables, constants, member function and other functionality.

② object :- An object is a real world entity that has attributes, behavior, and properties. It is referred to as an instances of the class.

Different object have different states or attributes and behaviours

~~*~~ Class Declaration consist of :-

① Modifiers :- These can be Public or default access.
② Class name :- Initial letter
③ Superclass :- A class can only extend (subclass) one parent.
④ Interfaces :- A class can implement more than one interface.
⑤ Body :- Body surrounded by braces { }.

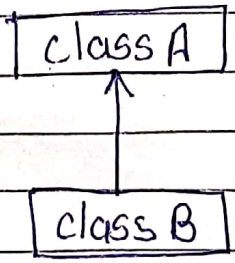
DATE
PAGE

③ Inheritance :- When one object acquires all the properties and behaviour of a parent object, It is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Inheritance is method in which one object acquires / inherits another object's properties, and inheritance also supports hierarchical classification. Inheritance represent the parent - child relationship.

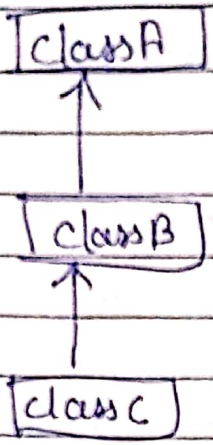
There are five types of inheritance Single, Multiple, Multilevel, hybrid, hierarchical.

① Single level Inheritance :- When a class inherits another class, It is known as a single inheritance. In the example given below, Dog class inherits the Animal class, So there is the single inheritance.

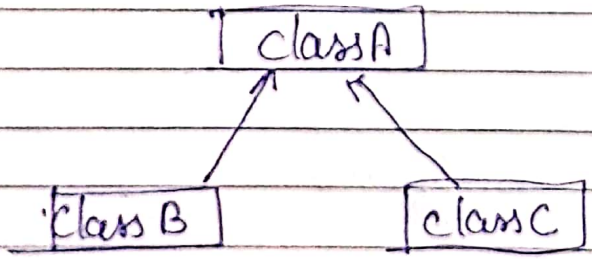


② Multilevel Inheritance :- When there is a chain of inheritance, It is known as multilevel inheritance. As you can see in the example given below, BabyDog class inherits the Dog class which again

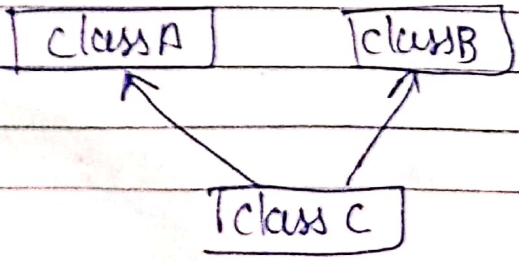
inherits the Animal class, so there is a multilevel inheritance



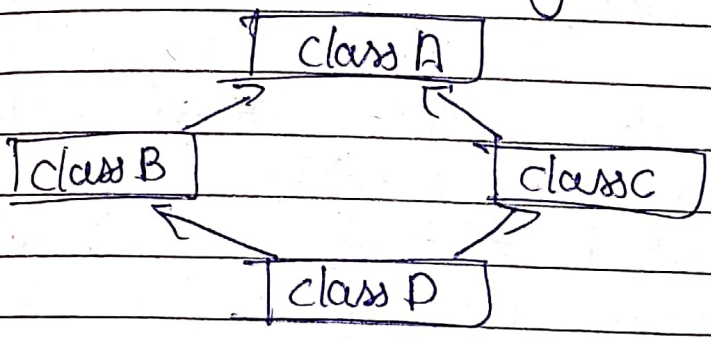
③ Hierarchical Inheritance :- When two or more classes inherits a single class, It is known as hierarchical inheritance. In the example given below, Dog and Cat classes inherits the Animal class, So there is hierarchical inheritance



④ Multiple Inheritance :- When one class inherits multiple classes, It is known as multiple inheritance.



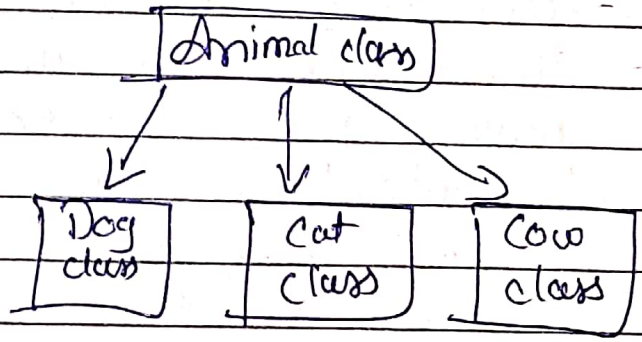
Hybrid Inheritance :- This is a Combination of Multiple and Hierarchical Inheritance. And this type of inheritance can only be achieved through interface.



Single Inheritance

```
class Animal
{
void eat()
{
Syst
```

Inheritance Example :-



④ Message passing :- Encourages object communication by allowing objects to request or give services by allowing objects to request or give services by calling methods or sending messages.

⑤ Dynamic binding :- Also known as late binding, this is the process of deciding which method to call when the program runs, not when the code is written

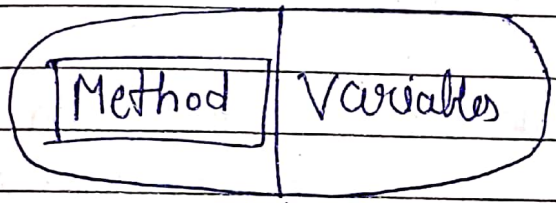
⑥ Polymorphism :- Polymorphism refers to many forms, or it is a process that performs a single action in different ways. It occurs when we have many classes related to each other by inheritance. For example :- A person at the same time can have different characteristic A man at the same time is a father, a husband and a employee. So the same person possesses different behaviours in different situation. This is called polymorphism.

Polymorphism in Java can be classified into two types :-

1. Static / Compile time Polymorphism
2. Dynamic / Runtime Polymorphism

DATE
PAGE

⑦ Encapsulation :- Encapsulation is defined as wrapping up data and information under a single unit. In OOP, Encapsulation is defined as binding together the data and the functions that manipulate them.
Consider a real life example of encapsulation, in a company, there are different section like the accounts section, finance section, sales section, etc.



Class

For example ~~me~~ capsul. A Capsule contain several type of medicines in single unit.

⑧ Abstraction :- Abstraction means displaying only essential information and hiding the details. Abstraction is a process which displays only the information needed and hides the unnecessary information. We can say that the main purpose of abstraction is data hiding.

Q) How OOP is different with POP?

OOP	POP
(1) ~~Object~~ OOPs stand for object oriented programming	Stand for Procedural oriented progeramming
(2) object oriented	Structure oriented
(3) Bottom up approach	top-down approach
(4) Program is divided into objects.	Program is divided into function.
(5) Inheritance property is used.	Inheritance is not allowed.
(6) It use access Specifier.	It doesn't use access specifier
(7) Encapsulation is used to hide the data	No data hiding
(8) Concept of Virtual function	No virtual function.

Flowchart :- The flowchart is a type of diagram graphical or symbolic that represents an algorithm or process.

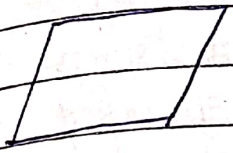
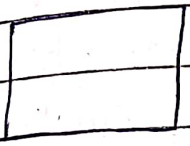
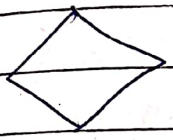
2. Each step in the process is represented by a different symbol and contains a short description of the process step.

3. The flowchart symbols are ~~it~~ linked together with arrows showing the process flow direction.

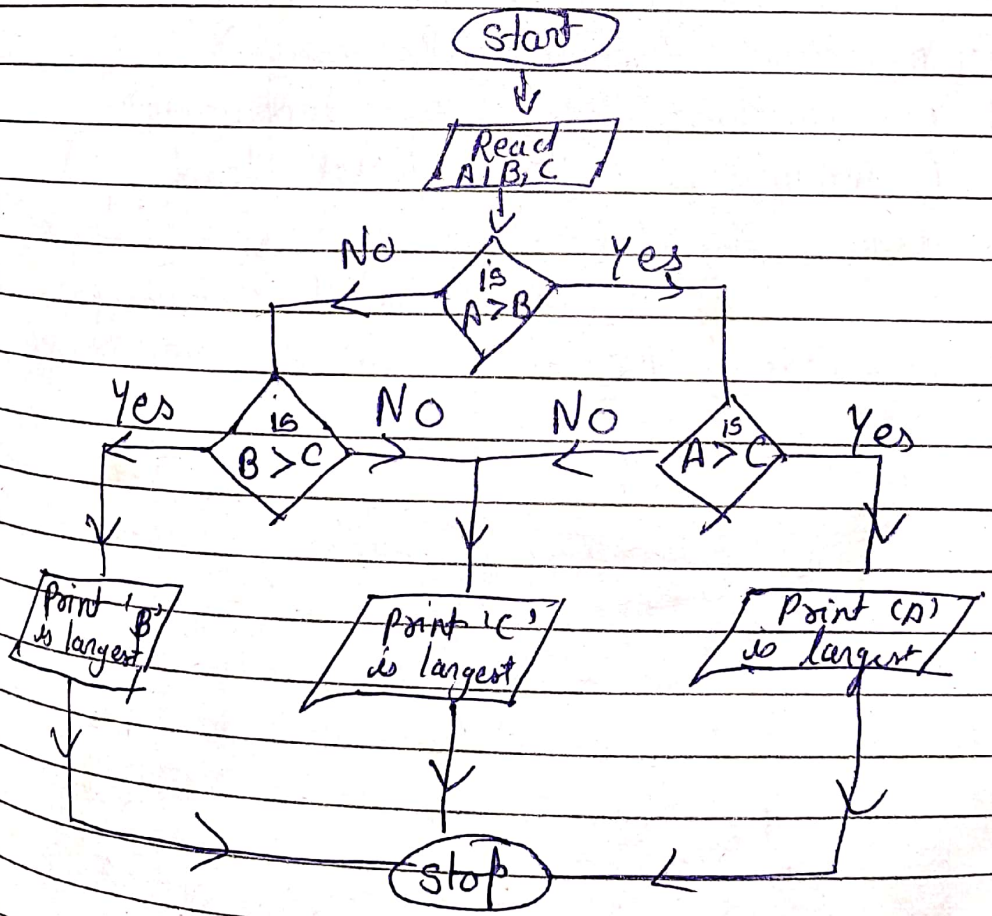
4. A flowchart typically shows the flow of data in a process.

5. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.

	Symbol	Name	Function
(1)	⬭	Start / end	An oval represents a start or end point.
(2)	→	Arrows	A line is a connector that shows relationship between the representative shapes

Symbol	Name	Function
③ 	Input / Output	A parallelogram represent input or output.
④ 	Process	A rectangle represents a process
⑤ 	Decision	A diamond indicates a decision.

Q:- Draw a flowchart to find the largest among three different numbers entered by the user.

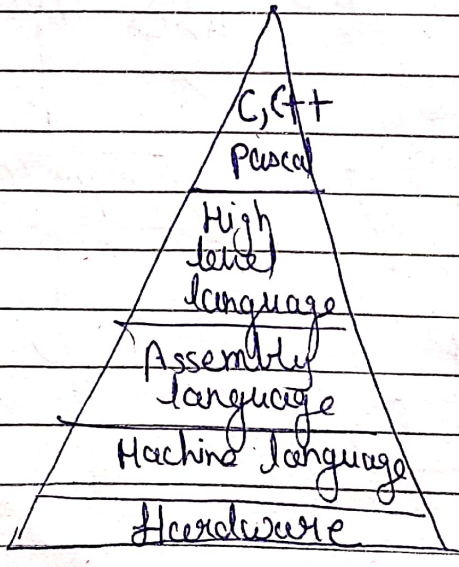


Programming language :- It is a set of symbols that instructs the hardware of computer to perform a specified task. There are set of rules to use these symbols called syntax. To perform a particular task the programmer writes a sequence of instructions, called the programs by using these symbols.

* Syntax
* Structure
* set of commands or instructions.

Categorization of Programming languages [Based on Generation]

* First Generation (Machine language)
* Second Generation (Assembly language)
* Third Generation (High level language)
* Fourth Generation (Domain Specific language and Database Query Language)
* Fifth Generation (Artifical Intelligence language)



First Generation language

Machine language :- Machine language is the lowest level programming language understood directly by a computer's hardware. It consists of binary code, typically represented by sequences of 0s and 1s, which correspond to specific instruction executed by the computer's CPU.

Advantages :- ① Efficiency ② low level control ③ Fast Execution ④ No translator is required.

Disadvantages :- ① Difficult to learn binary codes.

② Difficult to understand - both programs & where the error occurred.

③ Complexity ④ platform Dependent ⑤ Maintenance challenges.

Second Generation language

Assembly language :- Assembly language is a low level programming language that uses mnemonic codes and symbols to represent machine instructions, facilitating human readable programming close to the hardware level. Unlike machine language, which consists of binary and hexadecimal characters, assembly

languages are designed to be readable by humans.

Advantages of Assembly language:-

① Efficiency :- Direct control over hardware leads to highly optimized programs.

② low level Access :- Allow precise manipulation of system resources.

③ Performance :- Suitable for performance critical tasks.

Disadvantages of Assembly language :-

① Complexity :- Requires deep understanding of computer architecture.

② platform Dependence :- Not portable across different hardware architectures.

③ Error prone :- Manual memory management increases likelihood of bugs.

Difference Between Machine language and assembly language :-

Machine language	Assembly language
① Machine language is only understand by the computers	Assembly language is only understand by human beings not by the computers.
② In machine language data only represented with the help of binary format (0s and 1s) Hexadecimal and octadecimal	In assembly language data can be represented with the help of mnemonics such as Mov, Add, Sub, End etc.
③ Machine language is very difficult to understand by human beings	Assembly language is easy to understand by the human being as compare to machine language.
④ Modifications and error fixing cannot be done in machine language	Modification and error fixing can be done in assembly language.
⑤ Machine language is hardware dependent	Assembly language is the machine dependent and it is not portable

Third Generation language

High level language :- High level language is a type of programming language designed to provide a more user-friendly and abstracted interface for writing software.

It uses human-readable syntax and abstracted constructs, allowing programmers to focus on problem-solving rather than dealing with low level hardware details.

These languages typically include features such as variables, functions, and control structure, making it easier to write, read and maintain code.

Advantages :-

1. Use of English like words makes it a human understandable language.
2. Lesser number of line of code as compared to above 2 language
3. Readability
4. Portability

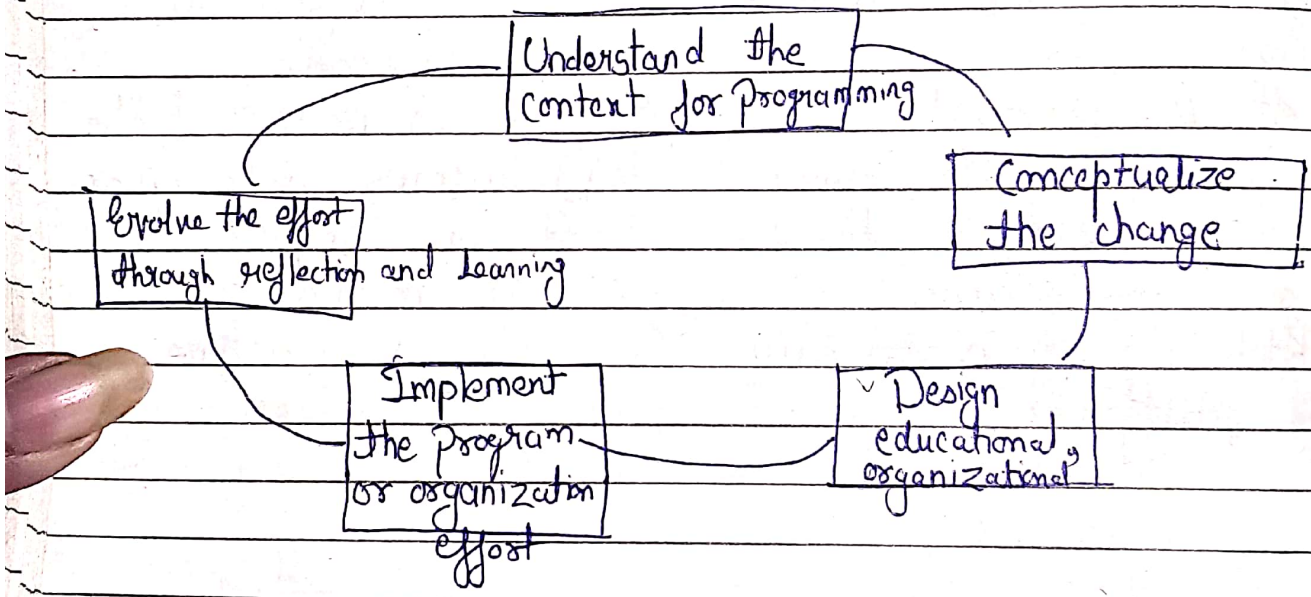
Disadvantages :-

1. Compiler / interpreter is needed
2. Different compilers are needed for different machines.

High level language	Vs	Low level language
① It is programmer friendly language		It is a machine friendly language
② High level language is less memory effitient.		Low level language is high memory efficient.
③ It is easy to understand		It is tough to understand
④ It is portable		It is non portable
⑤ Debugging is easy		Debugging is complex.
⑥ It is simple to maintain.		It is complex to maintain.
⑦ It can run on any platform.		It is machine dependent.

Program design :- Program design consists of the steps a Programmer should do before they start coding the program in a specific language.

These steps when Properly documented will make the complete Program easier for other programmers to maintain in the future.



① Understanding Requirement :- Gather and analyze Project needs and objectives.

② Decomposition and Algorithm Design :- Break down tasks into manageable components and design algorithms to solve them.

③ Data and interface design :- Design data structures and user interfaces

for effective data management and interaction.

(4) Testing, Refinement and Documentation :- Validate the design through testing, refine as needed and document the process for clarity and future references.

Procedural Oriented Programming

* The procedure oriented approach, large programs are divided into smaller program known as functions.

* In pop, a program is written as a sequence of procedures or function.

* In pop, each procedure (function) contain a series of instructions for performing a specific task.

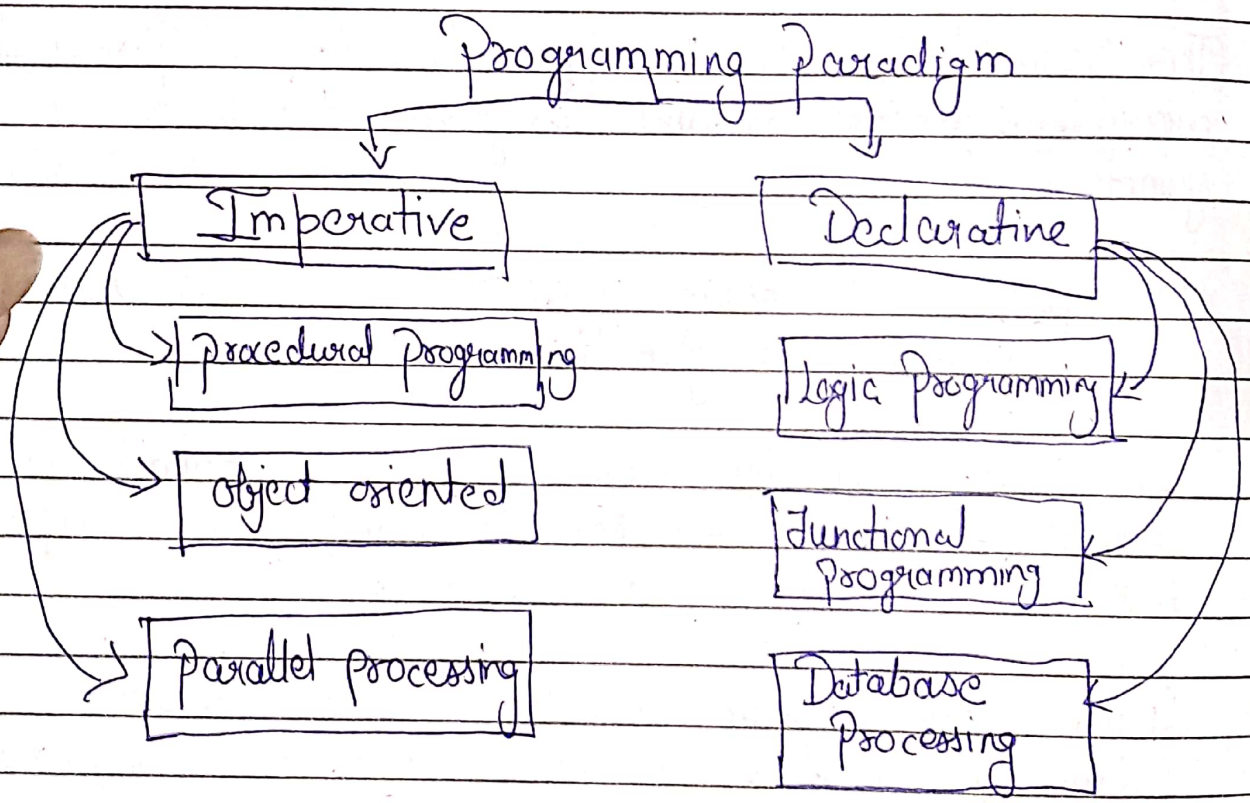
* During the program execution each procedure (function) can be called by the other procedures

* To call a procedure (function), we have to write function name only.

DATE
PAGE

Programming Paradigms :-

* Refers to classification of programming language based on their features and models.
* Languages can be classified into multiple paradigms.
* Based on execution model, syntax and grammar, code organization.



1. Imperative Programming Paradigm :-

* Is a programming paradigm that uses statements that change a program's state.
* It focuses close relation to machine architecture.

How to do that the Problem
problem ko kis tarika se solve karna hai

DATE
PAGE

* Based on von Neumann architecture.
* like C, Basic Fortran.

[a] Procedural programming paradigm :-

* This paradigm emphasizes on procedure in terms of under lying machine model.
* Code reusability.

[b] Object oriented programming :-

* Based on the concept of classes & objects.
* Methods and attributes (code and data).
* It can handle almost all kind of real-life problems which are today in scenario.
* like Simula, java, Python, ruby.

[c] Parallel Processing Approach :-

* Parallel Processing is the processing of program instructions by dividing them among multiple processors.
* A Parallel Processing is system passes many numbers of processor with the objective of running a program in less time by dividing them.
* NESL Programming language.

What to do this problem
problem K Sath Kya Karna

DATE
PAGE

Declarative Programming Paradigm :-

* A style of building the structure of computer programs that expresses the logic of a computation without describing its control flow.

* It just declares the result we want rather how it has be produced.

* This is the only difference between imperative (how to do) and declarative (what to do) programming Paradigms.

[9] Logic Programming Paradigm

* In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produce result

* Use the concept of AI, Machine learning and perception models

* puzzle problems

[b] Functional Programming Paradigm :-

* Based on mathematics
* The key principle of this paradigms is the execution of series of mathematical functions.
* Data is loosely coupled of function

[c] Database Programming approach :-

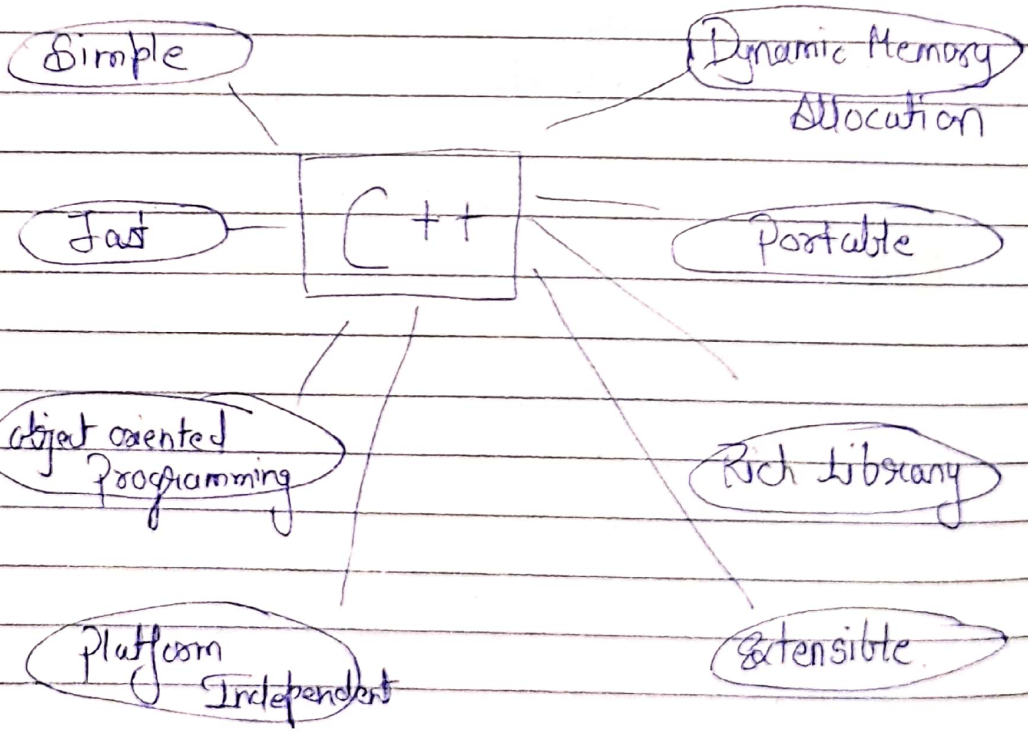
* Program statements are defined by data rather than hard-coding a series of steps.
* A Database program is the heart of a business information system and provides file creation, data entry, update, query and reporting function.
* AWK, SQL.

C++ language :-

* C++ is general-purpose programming language that was developed as an enhancement of the C language to include object oriented paradigm.

* C++ comes with a variety of features including class, objects, Inheritance, Data Encapsulation, Data Abstraction etc.

* Widely used in System/ Software development, game development, drivers, client-server applications and embedded firmware.



C++ Character Set

A character set is a defined list of characters that a programming language recognizes and uses to write programs. These characters include letters, digits and various special symbols that can be used in a C++ program.

The character set defines the valid characters that can be used for writing code,

including :

* Letters
* Digits
* Special characters
* White spaces
* Control characters

Types	character set
Uppercase Alphabets	A, B, C --Y, Z
Lowercase Alphabets	a, b, c --y. z
Digits	0, 1, 2, 3--9
Special Symbols	~ ! @ ? # % . ' ' $? ^ & * ( ) - + = ; -
White spaces	Single space, tab, new line

Tokens in C++

Tokens are the smallest units of a C++ program that the compiler can understand. They are the basic building block used to construct C++ programs. In C++, tokens include keywords, identifiers, literals, operators and punctuation

Keywords :-

* Reserved words in c++ that have a predefined meaning and cannot be used as identifiers (variable names, function names, etc)
* Examples :- int, float, if, else, while, return.

Identifiers :-

* Names given to various program elements such as variable, functions, arrays etc.
* Must start with a letter (Uppercase or lowercase) or an underscore (_) followed by letters, digits or underscores.
* Examples :- ~~My~~ myVariable, calculateSum, total_count

Constants :-

* fixed values that do not change during the execution of the program.
* Types include integer constants, floating-point constants, character constants and boolean constants.

Examples :- 42, 3.14, 'A', true.
const int MAX_VALUE = 100;

DATE
PAGE

Strings :-

* Sequence of characters enclosed in double quotes.
* Strings are used to represent text in a program.
* Example :- "Hello, World!"

Operators :-

* Symbols that perform operations on variables and values.
* Types of operators include arithmetic operators, relational operators, logical operators, Bitwise operators, assignment operators and more.
* Example : +, −, *, /, =, ==, &&, ||

Precedence & Associativity :-

		Category	Associativity	Precedence
Parenthesis	()	Function call	Left to right	Highest
	[]	Array subscript		
	→	Arrow		
	.	Dot		
Unari	++	Increment	Right to left	
	--	Decrement		
	!	logical NOT		13
	~	One's complement		
	(type)	Cast		
	&	Address of		
	*	Indirection		
	Size of	Size of		
M	*	Multiplication	Left to right	12
	/	Division		
	%	Modulus		
A	+	Addition	Right to left	11
	-	Subtraction		
Shift	<<	Left shift	left to right	10
	>>	Right Shift		
Relational	<	less than	Left to right	
	<=	less than or Equal to		8
	>	Greater than		
	>=	Greater than or Equal to		
Bitwise	&	Bitwise AND	left to right	7
	^	Bitwise XOR	left to right	6
	|	Bitwise OR	Right to left	5

logical &&	logical AND	left to Right	4
logical ||	logical OR	left to Right	3
conditional ? :	Conditional	Right to left	2
Assignment = += -= *= /= %= >>= <<= &= ^= |=	Assignment operators	Right to left	1
Comma ,	Comma	left to right	lowest 0

Operator Precedence in C++ :-

In C++, operator precedence specifies the order in which operations are performed within an expression. When an expression contain multiple operators, those with higher precedence are evaluted before those with lower precedence.

Operator Associativity in C++ :-

Operator associativity determines the order in which operands are grouped when multiple operators have the same precedence. There are two types of associativity.

Left-to-right associativity :- In expressions like a+b-c, the addition and subtraction operators are evaluated from left to right so, (a+b)-c is equivalent.

Right to left associativity :- Some operators, like the assignment operator =, have right-to-left associativity. For example, a = b = 4; assign the value of b to a.

Program Structure :-

OPERATORS PRECEDENCE

```
#include <iostream>
using namespace std;
int main()
{
    // evaluates 17 * 6 first
    int num1 = 5 - 17 * 6;
    // equivalent expression to num1
    int num2 = 5 - (17 * 6);
    // forcing compiler to evaluate 5-17 first
    int num3 = (5-17) * 6;
    cout << "num1 = " << num1 << endl;
    cout << "num2 = " << num2 << endl;
    cout << "num3 = " << num3 << endl;
    return 0;
}
```

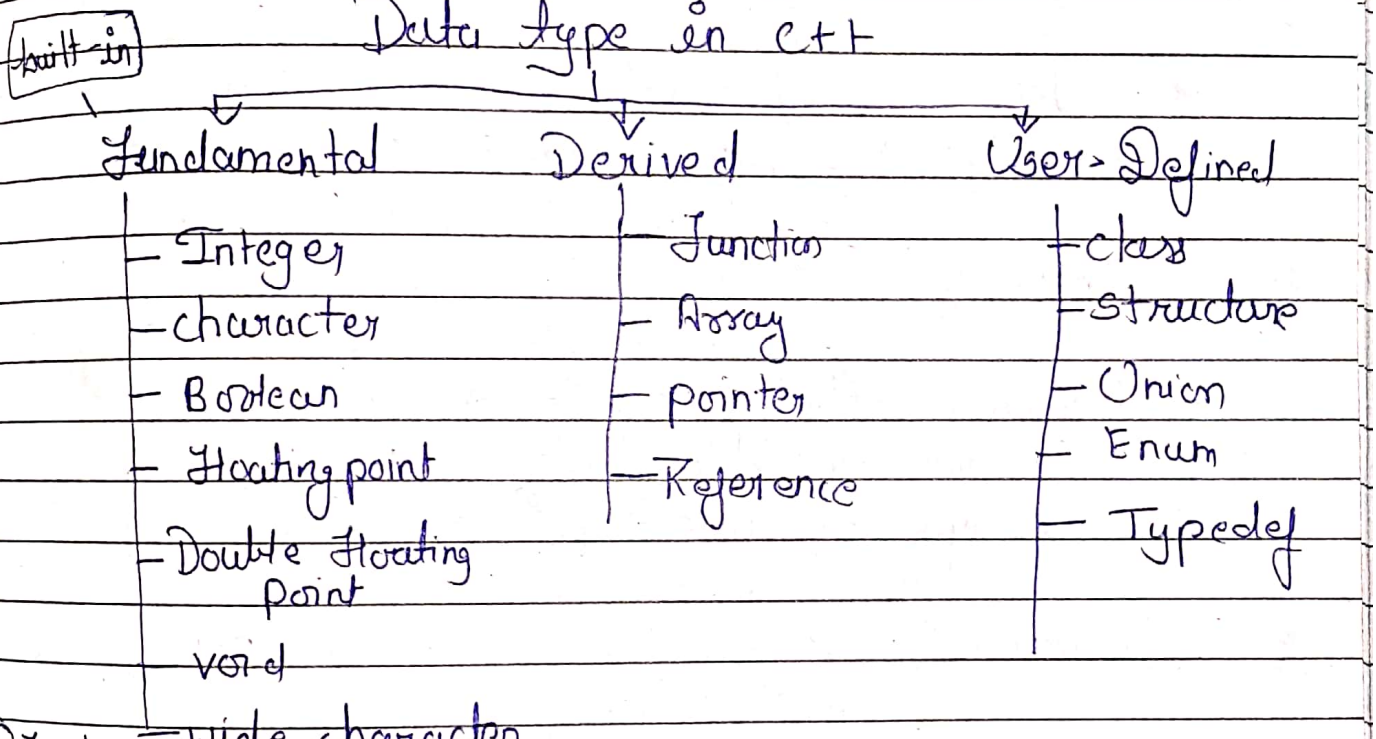
/output num1 = -97
num2 = -97
num3 = -72

OPERATORS ASSOCIATIVITY

```
#include <iostream>
using namespace std;
int main()
{
    int a=1, b=4;
// a-=6 is evaluated first
    b+=a-=6;
    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
}
```

Output
a=-5
b=-1

Data type :- Data types specify the type of data that a variable can hold. They are essential for allocating memory and performing operations on variables.



① Fundamental :-

* Integer (int) :- Used to store whole numbers
  - Example :- int age = 25;
* character (char) :- Used to store single characters.
  - Example :- char grade = 'A';
* Boolean (bool) :- Used to store true or false
  - Example :- bool is Student = true;
* Floating-point (float, double) :- Used to store decimal numbers.

  Ex :- float height = 5.9f;
  double distance = 123.456;
* Void (void) :- Used to specify that a function does not return a value.

- Example :- ~~void~~ void displayMessage() {...}

2. Derived Data types :-

1. Arrays :-

- Collection of elements of the same type.
- Example :- int numbers[5] = {1, 2, 3, 4, 5};

2. Pointers :-

- Variables that store the address of another variable.
- Example :- int * ptr = &age;

3. Function :-

- Block of code that performs a specific task.
- Example :-

```
int sum (int a, int b)
{
    return a+b;
}
```

3. User-defined Data type :-

① Structure :- (⊙ struct)

- Custom data type that groups variables of different types.

Example :-

```
struct Student
{
    string name;
    int age;
    float gpa;
};
Student student1 = {"John", 20, 3.75};
```

2. Unions (union)

- Similar to structures but share the same memory location for all its members.

Example :-

```
union Data
{
    int i;
    float f;
    char str[20];
};
Data data1;
```

3. Enumerations (enum)

- Used to define a variable that can hold only a set of predefined values.

Example :-

```
enum color { RED, GREEN, BLUE};
Color myColor = GREEN;
```

4. Classes (class)

- Blueprint for creating objects.
  Encapsulates data and function.

```
class Car {
public:
  string brand;
  string model;
  int year;
};
Car myCar;
myCar.brand = "Toyota";
myCar.model = "Corolla";
myCar.year = 2020;
```

Variables

A variable in c++ is a named storage location in memory that holds a value. This value can be changed during the program execution. Variables are used to store data that can be manipulated by the program.

Variables can be of different types depending on the types of data they store :-

int, float, boolean, char, String.

Syntax :-

```
data_type variable_name = value;
// defining single variable
```

Or

```
data_type variable_name1, variable_name2;
// defining multiple variable.
```

Here, ① data_type :- type of data that a variable can store

② Variable_name :- Name of the variable given by the user.

③ Value :- Value assigned to the variable by the user.

Program Structure :-

The C++ Program is written using a specific template structure. The structure of the program written in C++ language is as follows :-

Documentation
Link Section
Definition Section
Global Declaration section
Function definition section
Main Function
Skeleton of C program

Preprocessor Directives → `# include <iostream>` `# include <conio.h>` → Header Files

[Definition / Declaration Section]

`using namespace std;` → Namespace std

Namespace std ← `int` `main()` → Program Main Function (Entry point)

opening Brace ← {

[Body of Main function]

`return 0;` → Main function Return value

closing Brace ← }

[Function Definition Section

① Documentation Section :- The documentation section of a c++ program serves as an important tool for developers to explain the logic, purpose and structure of the code.

② Linking Section :- The linking section contains two parts :-
① Header Files ② Namespaces

③ ~~Global Declaration~~ Definition Section :-
It is used to declare some constants and assign them some value. Ex :- int integer.

④ Global Declaration Section :- Here, the variables and the class definitions which are going to be used in the program are declared to make them global.

⑤ Function Declaration Section :- It contains all the function which our main functions need.

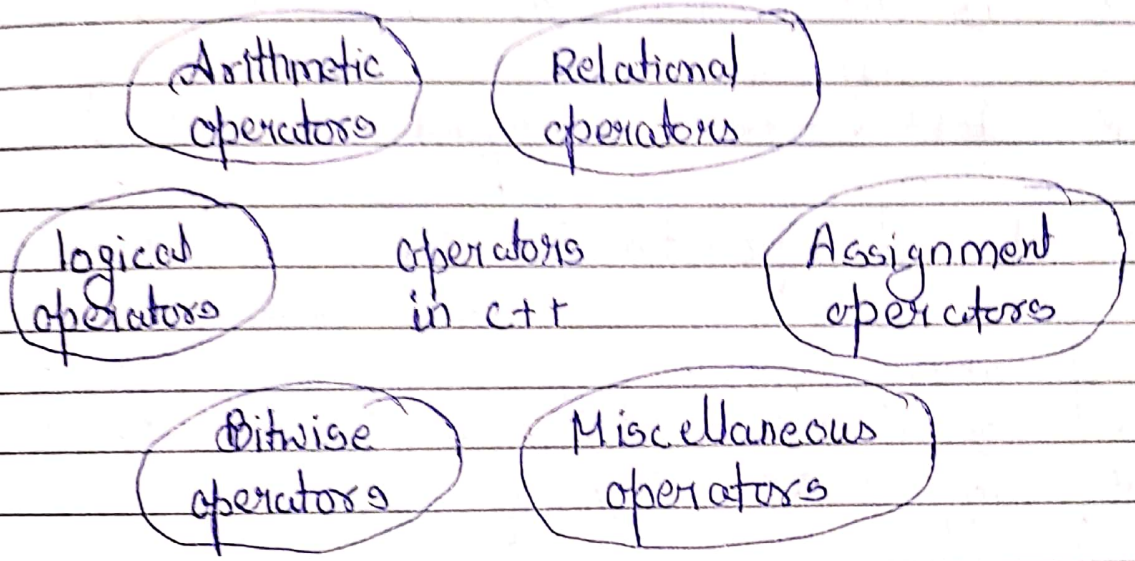
⑥ Main Function :- All the statements that are to be executed are written in the main function.

```
// Documentation Section
/* This is a C++
*/

// Linking Section
#include<iostream>
using namespace std;
// Definition Section
#define msg "Factorial\n"
typedef int Integer;
// Global Declaration Section
Integer num = 0, fact = 1, Storefactorial = 0;
// Function Section
Integer Integer factorial (Integer num)
{
    for (Integer i=1; i<=num; i++)
    {
        fact *= i;
    }
    // Return the factorial
    return fact;
}
// Main Function
Int Integer main()
{
    Integer Num = 5;
    storefactorial = factorial(Num);
    cout << msg;
    cout << Num << "!=" <<
        storefactorial << endl;
    return 0;
}
```

Factorial
5! = 120

Operators :- Operators are symbols that tell the compiler to perform specific mathematical, relational, or logical operations and produce a result. C++ provides a rich set of operators to manipulate variables.



Types of operators :-

1. Arithmetic operator
2. Relational operator
3. Increment or decrement
4. Conditional operator

① Arithmetic Operator :- These operators perform basic arithmetic operations.

* Addition (+) :- Adds two operands
  int sum = a+b
* Subtraction (-)
* Multiplication (*)
* division (/)
* Modules (%)

② Relational operators :- These operators compare two value and return a boolean result.

* equal to (==)
* Not equal to (!=)
* Greater than (>)
* less than (<)
* Greater than or equal to (>=) if (a>=b)
* less than or equal to (<=)

③ logical operators :- These operators are used to perform logical operations.

* logical AND (&&)
* logical OR (||) if (a||b)
* logical NOT (!) if (!a)

④ Assignment operators :- These operators assign value to variables.

* Assignment (=): // a=b
* Add and assign (+=) // a+=b // equivalent a = a+b
* Subtract and assign (-=) // a-=b // a=a-b

⑤ Increment and Decrement operators :-

These operators are used to increase or decrease the value of a variable by one.

* Increment (++)
* Decrement (--)

⑥ Conditional (Ternary) operator :- This operator is a shorthand for an if-else statement.

Expression :- An expression is a combination of variables constants and operators written according to the syntax of C language

Algebraic Expression	Expression
$a \times b - c$	a * b - c
$(m+n)(x+y)$	(m+n) * (x+y)
(ab/c)	a * b / c
$3x^2 + 2x + 1$	3 * x * x + 2 * x + 1
$(x/y) + c$	x/y + c

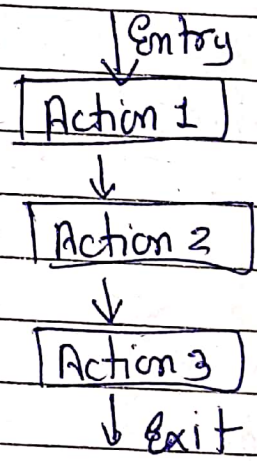
Expressions perform action based on a operator and one or two operands.
Operands can be constants, variables or function result.

Expressions perform specific actions, based on an operator with one or two operands.

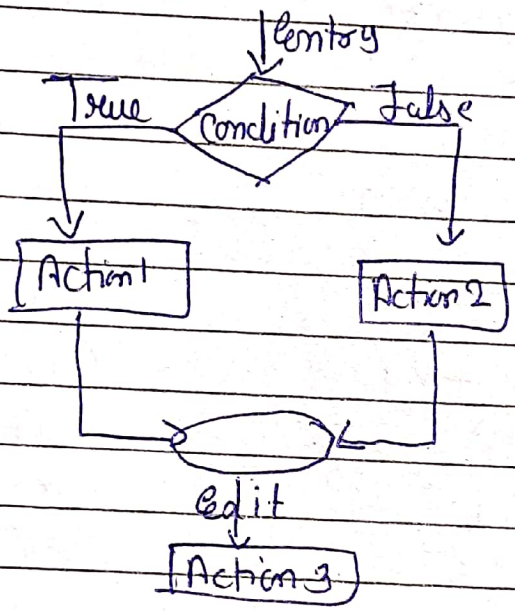
Control Structure in C++ & Statement :-

(1) Basic Control Statement
- (a) Sequence
- (b) Selection
- (c) Loop

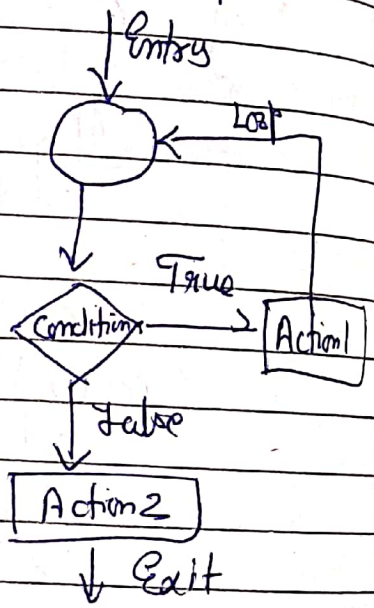
(a) Sequence

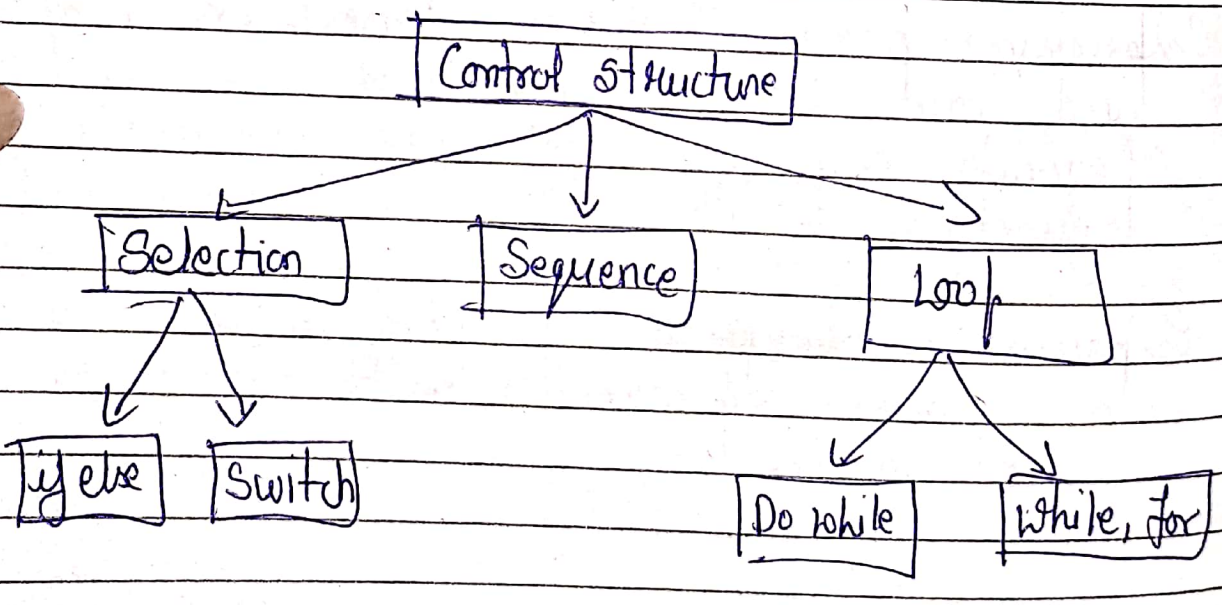


(b) Selection



(c) Loop

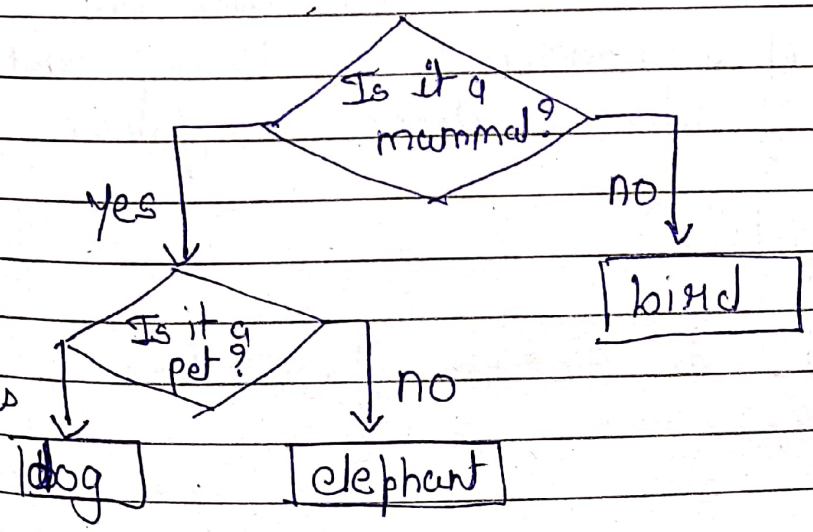




(1) Sequence Statement :- Sequence statement are the default mode of execution in a C++ program. They are executed one after another in the order they appear.

```
int a=10 ; // statement 1
int b=20 ; // statement 2
int sum = a+b ; // statement 3
cout << " Sum : " << sum; // statement 4
```

(2) Selection Statement :- Selection statement allow the program to choose different paths of execution based on certain conditions. The primary selection statement in C++ are if, if-else and switch.



(i) If statement :- The if statement executes a block of code if a specified condition is true.

```
if (this condition is true)
{
    // execute this code
}
```

Ex :-
```
int number = 10;
if (number > 0)
{
    cout << "The number is positive";
}
```

(ii) if-else statement :- The if-else statement executes one block of code if a condition is true, and another block if the condition is false.

```
if (Condition)
{
    // code to execute if Condition is true
}
else
{
    // Code to execute if Condition is false
}
```

(iii) if :- Switch statement :- The switch statement in C++ is used to execute the different blocks of statement based on the value of the given expression.

```
char grade = 'B';
switch (grade)
{
  case 'A':
    cout << "Excellent!";
    break;
  case 'B':
    cout << "Good!";
    break;
  case 'C':
    Cout << "Fair";
  break;
  default:
    cout << "Invalid grade";
}
```

(2) Looping statement :- Looping statement allow the execution of a block of code multiple times. The primary looping statements in C++ are for loop and while loop.

(i) for loop :- The for loop is used when the number of iterations is known beforehand. It consists of three parts :- initialization, condition and increment / decrement.

```
for (int i=1; i<=5; i++)
  {
    cout << i << " ";
  }
```

(ii) while loop :- It is used when the number of iteration is not known beforehand. It continues to execute as long as the specified condition is true.

```
while (Condition)
{
// code to execute
}
```

```
int i = 1;
while (i<=5)
  {
    cout << i << " ";
    i++;
  }
```

(iii) Do-While loop :- The do while loop is similar to the while loop, but it guarantees that the loop's body will execute at least once before the condition is tested.

```
do
  {
    // code to execute
  } while (Condition);
```

```
int i=1;
do
{
  cout << i << " ";
  i++;
} while (i<=5)
```

I/O Operations :-

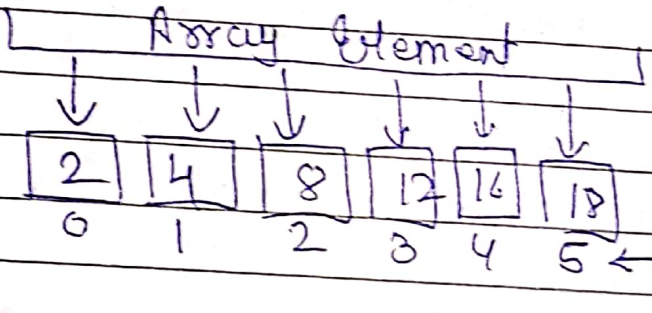
or input/output operations, are the processes that involve reading or writing data to and from external sources. These sources can be files, databases or network sockets.

Here are some examples of I/O operations :-

* Uploading a file to a web application :-
  - The file is stored on the server's disk through I/O operations.
* Reading data from a file :- In Node.js the fs module can be used to read data from a file.

Array :- An Array is a collection of elements of the same data type stored in contiguous memory locations. Arrays are used to store multiple values in a single variable, instead of declaring separate variable for each value.

Array in C++



Declaration and Initialization :-

* Declaration :-
  - Syntax : data-type array-name [array-size];
  - Example : int numbers [5];
* Initialization :-
  - Arrays can be initialized at the time of declaration.
  - Syntax : data-type array-name [array size] = { value 1, value 2 - - valueN};
  - Example : int number [5] = {1, 2, 3, 4, 5};
* Implicit Size :-
  - If the size of the array is not specified, it can be determined by the number of initializers.

Example :- int number [ ] = {1, 2, 3, 4, 5};

2- Dimensional Array :- A 2D array can be ~~throw~~ thought of as a table of elements, arranged in rows and columns. Each element in a 2D array can be accessed using two indices: one for the row and one for the column.

1D Array :-

C	O	D	I	N	G	E	← Single row of element
0	1	2	3	4	5	6	

2D Array :-

i \ j	0	1	2	← column
0	A	A	A	} array element
1	B	B	B	
2	C	C	C	

col 0 col 1 col 2

row 0
row 1
row 2

Declaration and Initialization :-

- Dot Declaration :-

Syntax :- data_type array_name [rows] [columns];

Example :- int matrix [2] [3];

* Initialization :-

- 2D array can be initialized at the time of declaration.
- Syntax :- data-type array-name [rows] [columns] = { { value 1, value 2 ... }, { value 3, value 4 } };

Example :- int matrix [2][3] =

{ {1, 2, 3} { 4, 5, 6} }.

```
int matrix [2][3] =
{
    {1, 2, 3},
    {4, 5, 6}
};
```

Function :- A function is a self-contained block of code designed to accomplish a specific task. Once a function is defined, it can be called whenever needed and it can be called multiple times within a program.

Why use functions :-

* Reusability * Modularity * Maintainability * Code Readability.

Function Declaration:- A function must be declared before it is used.

- Syntax:- return_type Function_name (parameter_list);

```
int findMax (int a, int b, int c);
#include <iostream>
using namespace std;
// Function definition
int findMax (int a, int b, int c)
{
    int max = a;
    if (b > max)
        max = b;
    if (c > max)
        max = c;
    return max;
}
int main ()
{
    int num1, num2, num3;
    cout << "Enter three num";
    cin >> num1 >> num2 >> num3;
    // Function call
    int maxNumber = findMax (num1, num2, num3)
    cout << "The maximum num is" <<
        maxNumber << endl;
    return 0;
}
```