Chameli Devi Group of Institutions, Indore
Department of ESH
BT205 Basic Computer Engineering
B. Tech, CSE and IT (II Semester)
Unit -2

……

Syllabus: Introduction to Algorithms, Complexities and Flowchart, Introduction to Programming, Categories of Programming Languages, Program Design, Programming Paradigms, Characteristics or Concepts of OOP, Procedure Oriented Programming VS object-oriented Programming.
Introduction to C++: Character Set, Tokens, Precedence and Associativity, Program Structure, Data Types, Variables, Operators, Expressions, Statements and control structures, I/O operations, Array, Functions.

Unit Objective: To familiarize the students with basic concepts of algorithm, flowchart, programming languages and concepts of C++ programming language.

Unit Outcome: Student should be able to write algorithm, draw flowchart for a given problem and also to acquire knowledge of C++ programming basic concepts.

……..

Algorithm

An algorithm is a step-by-step procedure that defines a set of instructions that must be carried out in a specific order to produce the desired result.
Algorithms are generally developed independently of underlying languages, which means that an algorithm can be implemented in more than one programming language.
An algorithm is a set of commands that must be followed for a computer to perform calculations or other problem-solving operations.
According to its formal definition, an algorithm is a finite set of instructions carried out in a specific order to perform a particular task.
It is not the entire program or code; it is simple logic to a problem represented as an informal description in the form of a flowchart or pseudocode.

Characteristics of Algorithm

Input: An algorithm requires some input values. An algorithm can be given a value as input.
Output: At the end of an algorithm, you will have one or more outcomes.
Unambiguity: A perfect algorithm is defined as unambiguous, which means that its instructions should be clear and straightforward.
Finiteness: An algorithm must be finite. Finiteness in this context means that the algorithm should have a limited number of instructions, i.e., the instructions should be countable.
Effectiveness: Because each instruction in an algorithm affects the overall process, it should be adequate.
Language independence: An algorithm must be language-independent, which means that its instructions can be implemented in any language and produce the same results.

Example of Algorithm

Problem: Create an algorithm that multiplies two numbers and displays the output.
Solution:
Step 1 − Start
Step 2 − declare three integers x, y & z

Step 3 − define values of x & y
Step 4 − multiply values of x & y
Step 5 − store result of step 4 to z
Step 6 − print z
Step 7 − Stop

Flowchart

Flowcharts are the graphical representation of the data or the algorithm for a better understanding of the code visually.

It displays step-by-step solutions to a problem, algorithm, or process.

It is a pictorial way of representing steps that are preferred by most beginner-level programmers to understand algorithms, thus it contributes to troubleshooting the issues in the algorithm.

A flowchart is a picture of pre-defined shapes that indicates the process flow in a sequential manner.

Since a flowchart is a pictorial representation of a process or algorithm, it's easy to interpret and understand the process.

Uses of Flowchart

It is a pictorial representation of an algorithm that increases the readability of the program.

Complex programs can be drawn in a simple way using a flowchart.

It helps team members get an insight into the process and use this knowledge to collect data, detect problems, develop software, etc.

A flowchart is a basic step for designing a new process or add extra features.

Communication with other people becomes easy by drawing flowcharts and sharing them.

When to use Flowchart

It is mostly used when the programmers make projects. As a flowchart is a basic step to make the design of projects pictorially, it is preferred by many.

When the flowcharts of a process are drawn, the programmer understands the non-useful parts of the process. So, flowcharts are used to separate useful logic from the unwanted parts.

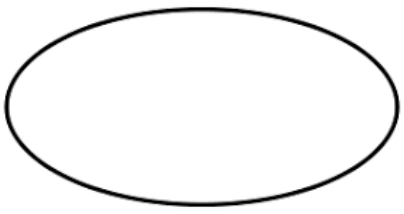
Since the rules and procedure of drawing a flowchart are universal, flowchart serves as a communication channel to the people who are working on the same project for better understanding.

Optimizing a process becomes easier with flowcharts. The efficiency of code is improved with the flowchart drawing.

Symbols to create flowchart

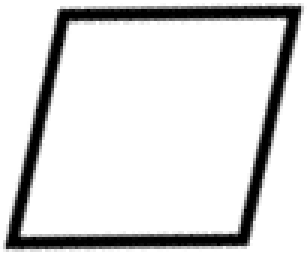
Terminal: This box is of an oval shape which is used to **indicate the start or end of the program**.

Every flowchart diagram has this oval shape that depicts the start of an algorithm and another oval shape that depicts the end of an algorithm.



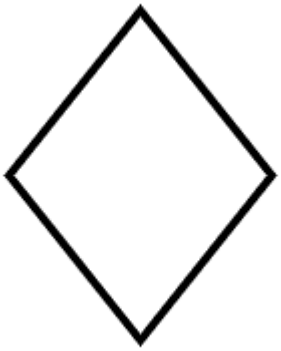
Data: This is a parallelogram-shaped box inside which the inputs or outputs are written.

This basically depicts the information that is entering the system or algorithm and the information that is leaving the system or algorithm.



Process: This is a rectangular box inside which a programmer writes the main course of action of the algorithm or the main logic of the program.
This is the crux of the flowchart as the main processing codes is written inside this box.

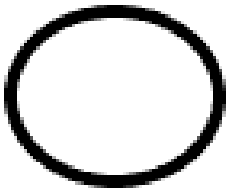
Decision: This is a rhombus-shaped box, control statements like if, or condition like a > 0, etc are written inside this box.
There are 2 paths from this one which is “yes” and the other one is “no”.



Flow: This arrow line represents the flow of the algorithm or process. It represents the direction of the process flow.

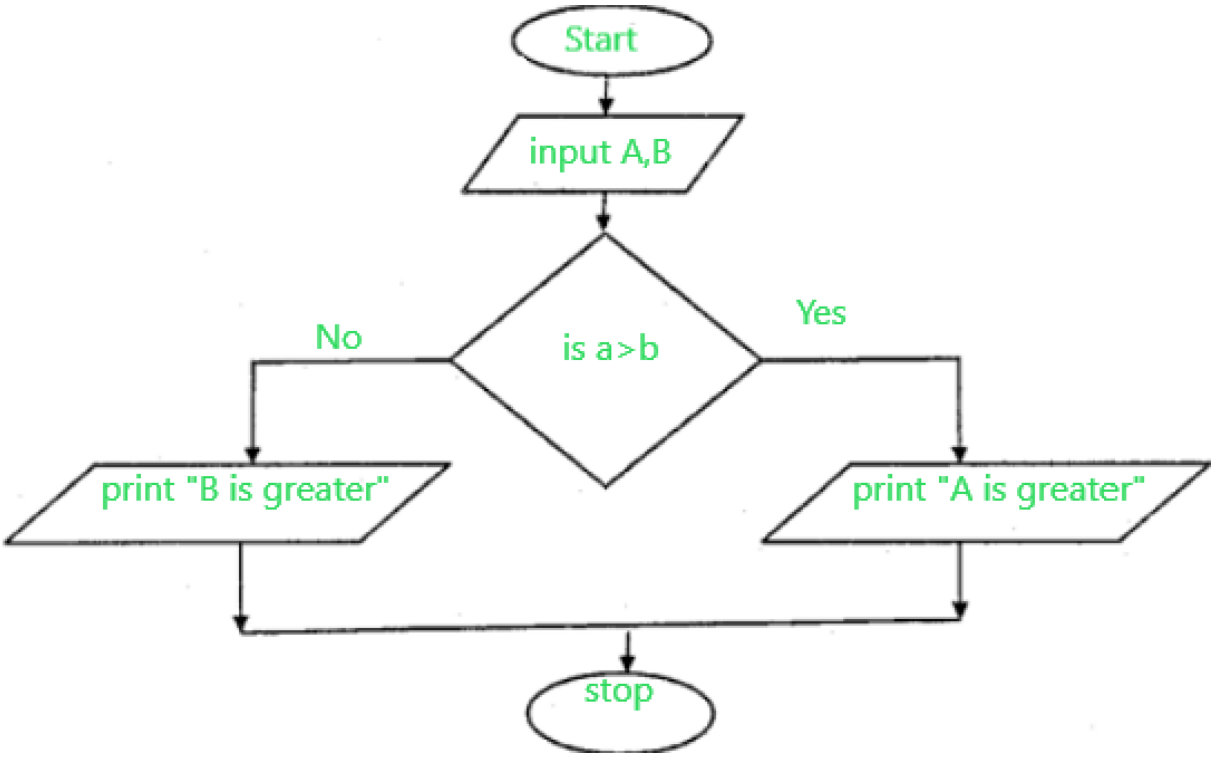
One Page Reference: This circular figure is used to depict that the flowchart is in continuation with the further steps.
This figure comes into use when the space is less and the flowchart is long.

Any numerical symbol is present inside this circle and that same numerical symbol will be depicted before the continuation to make the user understand the continuation.

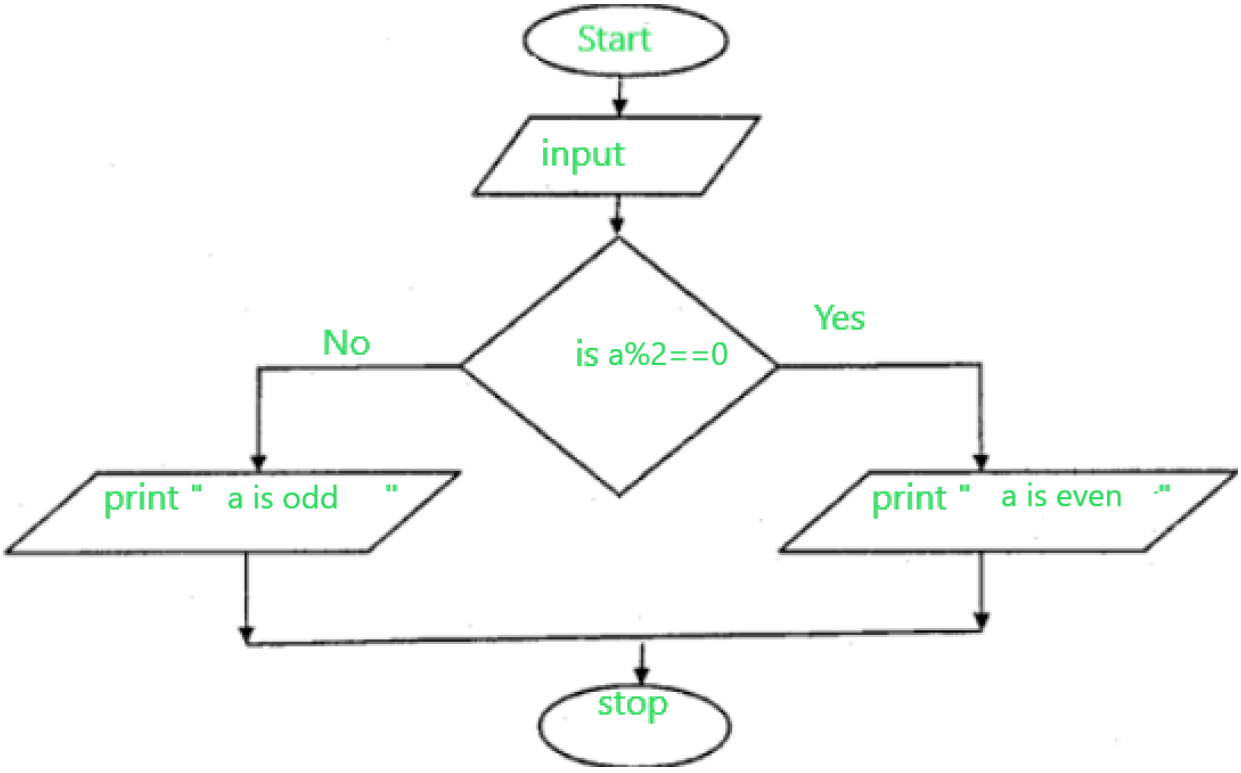


Example of Flowchart

Draw a flowchart to find the greatest number among the 2 numbers.



Draw a flowchart to check whether the input number is odd or even



Complexity

Complexity is a measure of an algorithm. It determines the amount of time and space necessary to execute the algorithm. The computational problem can be solved by different algorithms, Calculation complexity helps us to select efficient algorithm.

Time Complexity: Time require to solve a computational problem is called Time complexity of the algorithm.

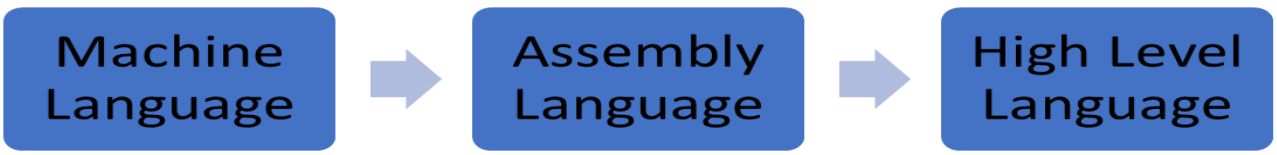
Space Complexity: The space requirement of a computational problem is called Space complexity.

Programming Language

A programming language is a set of symbols, grammars and rules with the help of which one is able to translate algorithms to programs that will be executed by the computer.

The programmer communicates with a machine using programming languages.

Most of the programs have a highly structured set of rules.



Machine Language

Machine language is a collection of binary digits or bits that the computer reads and interprets.

Machine language is the only language a computer is capable of understanding.

Machine level language is a language that supports the machine side of the programming or does not provide human side of the programming.

It consists of (binary) zeros and ones. Each instruction in a program is represented by a numeric code, and

numerical addresses are used throughout the program to refer to memory locations in the computer's memory.
Microcode allows for the expression of some of the more powerful machine level instructions in terms of a set of basic machine instructions.

Assembly Language

An assembly language is a type of low-level programming language that is intended to communicate directly with a computer's hardware.
Unlike machine language, which consists of binary and hexadecimal characters, assembly languages are designed to be readable by humans.
A type of programming language that translates high-level languages into machine language.
Bridge between software programs and their underlying hardware platforms.
Relies on language syntax, labels, operators, and directives to convert code into usable machine instruction.
In present scenario, they are rarely written directly, although they are still used in some niche applications such as when performance requirements are particularly high.

High Level Language

High level language is a language that supports the human and the application sides of the programming.
A language is a machine independent way to specify the sequence of operations necessary to accomplish a task.
A line in a high-level language can execute powerful operations, and correspond to tens, or hundreds, of instructions at the machine level.
Consequently, more programming is now done in high level languages.
Examples of high-level languages are BASIC, FORTRAN etc

Language Processors

Every program must be translated into a machine language that the computer can understand.
This translation is performed by Language Processors.

Language Processor: Compiler

The language processor that reads the complete source program written in high-level language as a whole in one go and translates it into an equivalent program in machine language is called a Compiler.
Example: C, C++, C#, Java.
In a compiler, the source code is translated to object code successfully if it is free of errors.
The compiler specifies the errors at the end of the compilation with line numbers when there are any errors in the source code.
The errors must be removed before the compiler can successfully recompile the source code again the object program can be executed number of times without translating it again.

Language Processor: Assembler

The Assembler is used to translate the program written in Assembly language into machine code.
The source program is an input of an assembler that contains assembly language instructions.
The output generated by the assembler is the object code or machine code understandable by the computer.
Assembler is basically the first interface that is able to communicate humans with the machine.

We need an Assembler to fill the gap between human and machine so that they can communicate with each other.
Code written in assembly language is some sort of mnemonics(instructions) like ADD, MUL, MUX, SUB, DIV, MOV and so on.
These mnemonics also depend upon the architecture of the machine.

Language Processor: Interpreter
The translation of a single statement of the source program into machine code is done by a language processor and executes immediately before moving on to the next line is called an interpreter.
If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message.
The interpreter moves on to the next line for execution only after the removal of the error.
An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code.
Example: Perl, Python and MATLAB

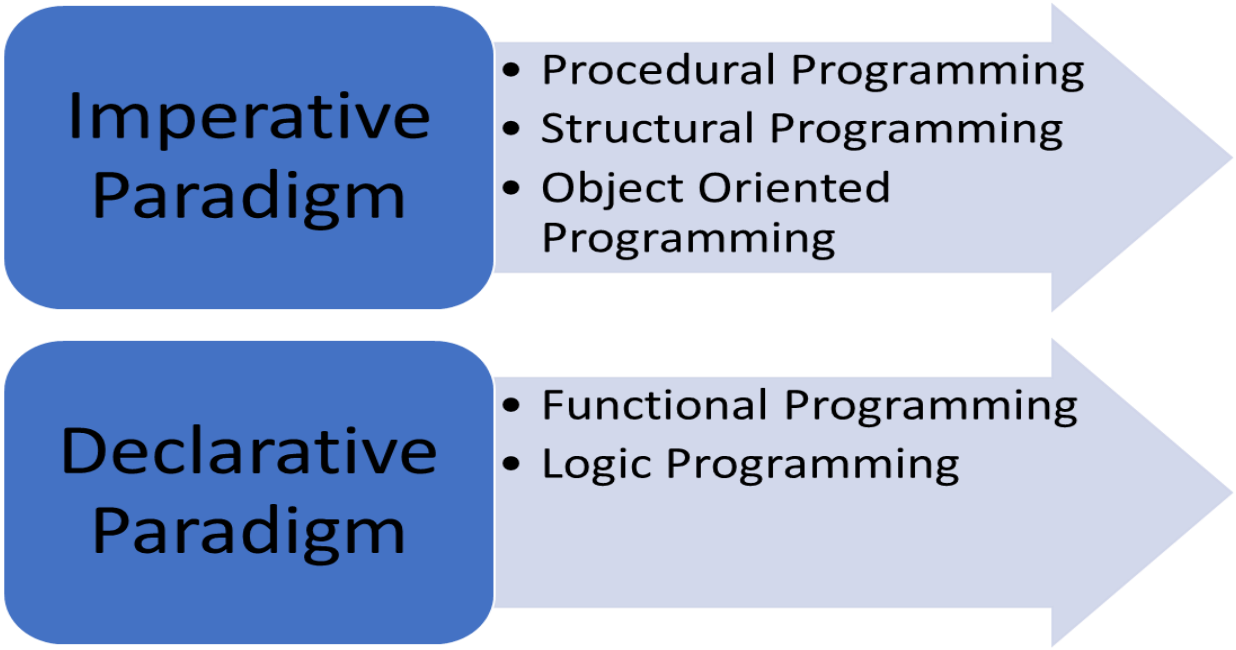
Program Design
Programming design deals with the analysis, design and implementation of programs.

Program Design: Top-Down Approach
An approach to design algorithms in which a bigger problem is broken down into smaller parts.
It uses the decomposition approach.
This approach is generally used by structured programming languages such as C, COBOL, FORTRAN.
The drawback of using the top-down approach is that it may have redundancy since every part of the code is developed separately.
Less interaction and communication between the modules in this approach.
The implementation of algorithm using top-down approach depends on the programming language and platform.
Generally used with documentation of module and debugging code.

Program Design: Bottom-Up Approach
Bottom-Up Approach is one in which the smaller problems are solved, and then these solved problems are integrated to find the solution to a bigger problem.
It uses composition approach.
It requires a significant amount of communication among different modules.
It is generally used with object-oriented programming paradigm such as C++, Java, and Python.
Data encapsulation and data hiding is also implemented in this approach.
The bottom-up approach is generally used in testing modules.

Programming Paradigm
Programming paradigm is all about the writing style and organizing the program code in a specific way.
Each paradigm focuses on a specific way to organize the program code.
A programming language is essentially a problem-solving tool.
For each problem there can be many solutions. Further, each solution can adapt a different approach in

providing solution to the problem.
For example, C language follows Procedural Programming Paradigm whereas C++, Python and JAVA are said to be Object Oriented Programming Paradigm.
In the context of programming languages, the term paradigm means set of design principles that defines the program structure.



Imperative Paradigm
Imperative paradigm is said to be command driven. The program code directs the program execution as sequence of statements executed one by one.
The imperative style program consists of set of program statements. Each statement directs the computer to perform specific task.
The programmer has to elaborate each statement in details. Each statement directs what is to be done and how it is to be done.
The execution of the program statements is decided by the control flow statements and the program flow can be directed as per the program logic.
The imperative paradigm programming languages include FORTRAN, ALGOL, PASCAL and Basic.

Declarative Paradigm
Declarative paradigm is a programming paradigm that is focused on the logic of the program and the end result.
The control flow is not the important element of the program.
The main focus of the declarative style of programming is achieving the end result.

Object Oriented Programming
Object-oriented programming uses objects in programming.
Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc. in programming.
The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOP Characteristics:

Object

An object is a real-world entity that has attributes, behavior, and properties. It is referred to as an instance of the class. It contains member functions & variables that we have defined in the class.

It occupies space in the memory. Different objects have different states or attributes, and behaviors.

Class

A class is a blueprint or template of an object. It is a user-defined data type.

Inside a class, we define variables, constants, member functions, and other functionality. it binds data and functions together in a single unit. Classes are not considered as a data structure. It is a logical entity. It is the best example of data binding. A class can exist without an object but vice-versa is not possible.

Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc.

So here, the Car is the class, and wheels, speed limits, and mileage are their properties.

Abstraction

The concept allows us to hide the implementation from the user but shows only essential information to the user.

Using the concept developer can easily make changes and added over time.

Advantages of Abstraction

It reduces complexity.

It avoids delicacy.

Eases the burden of maintenance

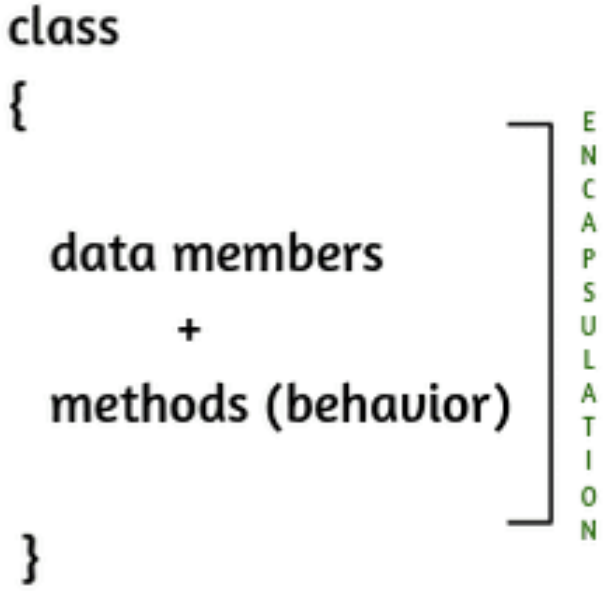
Increase security and confidentially.

Encapsulation

Encapsulation is a mechanism that allows us to bind data and functions of a class into an entity.

It protects data and functions from outside interference and misuse.

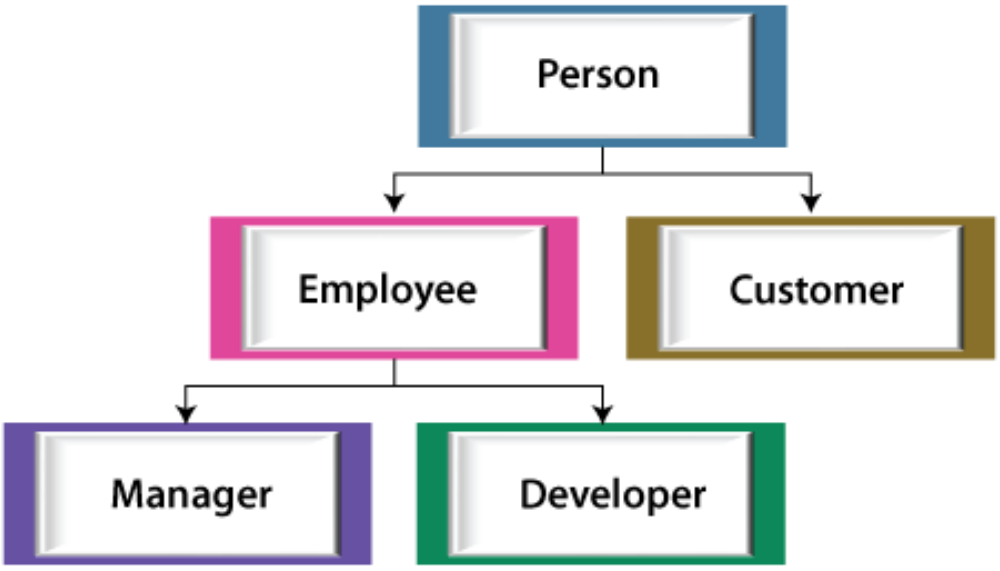
Therefore, it also provides security. A class is the best example of encapsulation.



Inheritance

The concept allows us to inherit or acquire the properties of an existing class (parent class) into a newly created class (child class).

It is known as inheritance. It provides code reusability.

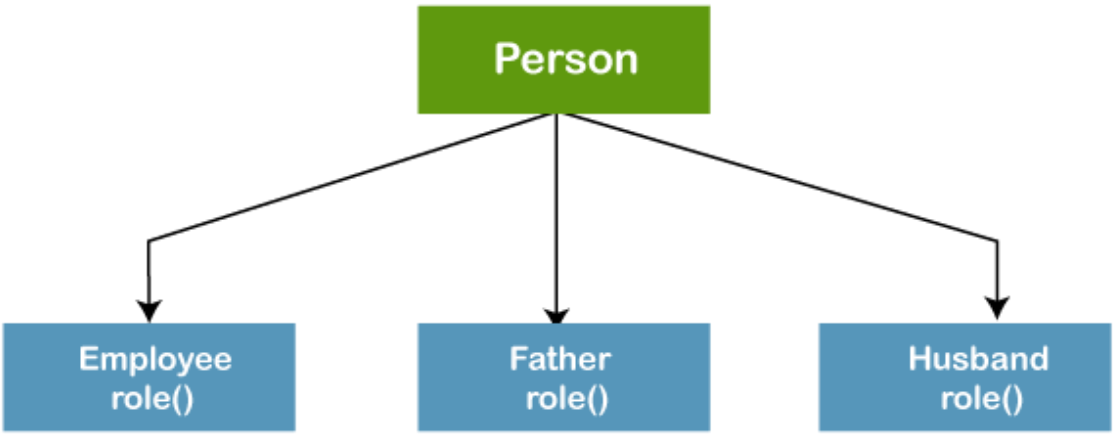


Polymorphism

The word polymorphism is derived from the two words i.e. ploy and morphs.

Poly means many and morphs means forms.

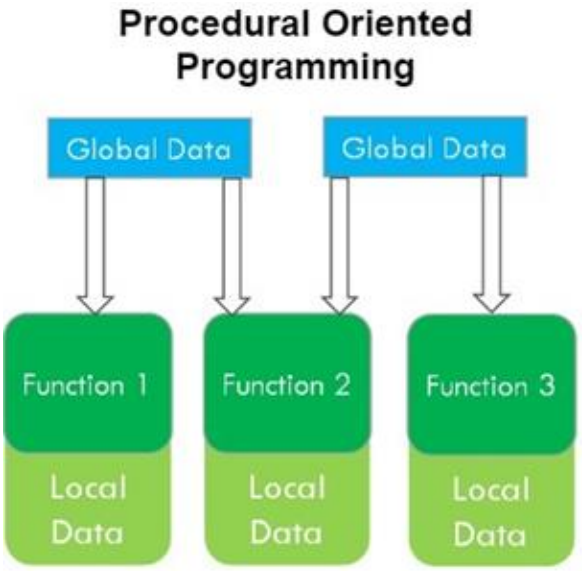
It allows us to create methods with the same name but different method signatures.

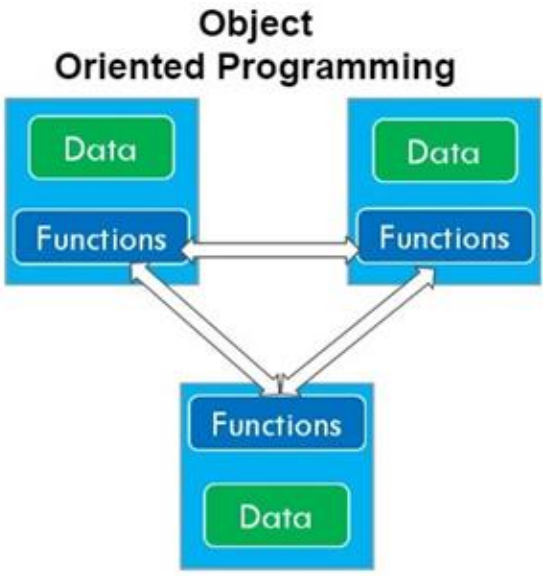


POP vs OOP

Factor	**OOP**	**POP**
Definition	OOP stands for Object Oriented Programing.	POP stands for Procedural Oriented Programming.

Approach	OOP follows bottom-up approach.	POP follows top-down approach.
Division	A program is divided to objects and their interactions.	A program is divided into functions and they interact.
Inheritance supported	Inheritance is supported.	Inheritance is not supported.
Access control	Access control is supported via access modifiers.	No access modifiers are supported.
Data Hiding	Encapsulation is used to hide data.	No data hiding present. Data is globally accessible.
Example	C++, Java	C, Pascal





C++ Introduction

C++ is a high-level, general-purpose programming language that was developed as an enhancement of the C language to include object-oriented paradigm. It is an imperative and a compiled language.

It was developed by Bjarne Stroustrup at Bell Labs in 1983 as an extension of the C programming language. C++ provides a rich set of libraries and features for high-level application programming, making it a popular choice for developing desktop applications, video games, and other complex applications.

C++ Character Set

The character set is a combination of English language comprising of the Alphabets and the White spaces and some symbols from the mathematics including the Digits and the Special symbols. C++ character set means the characters and the symbols that are understandable and acceptable by the C++ Program. These are grouped to create and give the commands, expressions, words, c-statements, and some of the other

tokens for the C++ Language.

C++ Token

A token is the tiniest element of a 'C++' program that is meaningful to the compiler.

TOKEN: Keyword, Identifiers, Constants, Strings, Special symbols, Operators

Keyword: In C++, keywords are reserved words that have a specific meaning in the language and cannot be used as identifiers.

Identifier: An identifier is a name given to a variable, function, or another object in the code. Identifiers consists of letters, digits, and underscores.

Below are the rules must be followed when choosing an identifier:

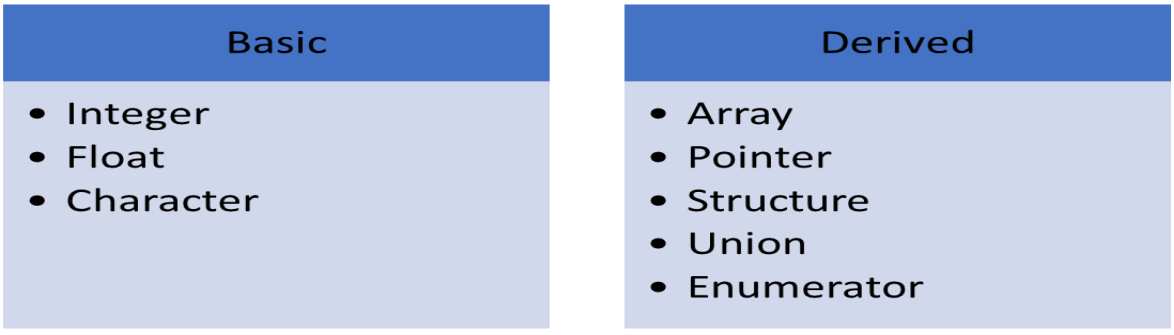
- The first character must be a letter or an underscore.
- Identifiers cannot be the same as a keyword.
- Identifiers cannot contain any spaces or special characters except for the underscore.
- Identifiers are case-sensitive, meaning that a variable named "myVariable" is different from a variable named "myvariable".

Constant: In C++, a constant is a value that cannot be changed during the execution of the program. Constants often represent fixed values used frequently in the code, such as the value of pi or the maximum size of an array.

String: In C++, a string is a sequence of characters that represents text. Strings are commonly used in C++ programs to store and manipulate text data. To use strings in C++, you must include the <string> header file at the beginning of your program.

C++ Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.



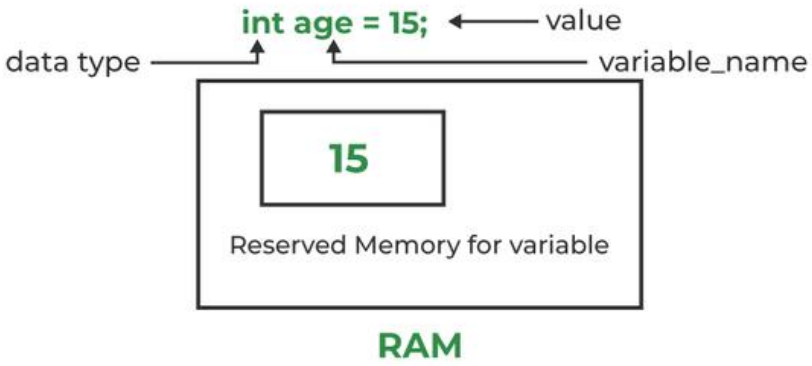
Basic Data Type: The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals. The memory size of basic data types may change according to 32- or 64-bit operating system.

Type	Size (Bytes)	Range
Int	2	-32768 to 32767
Signed int	2	Same as int

Unsigned int	2	0 to 65535
Char	1	-128 to 127
Signed char	1	Same as char
Unsigned char	1	0 to 255
Float	4	3.4x10-38 to 3.4x1038-1 (7 digit of precision)
Double	8	1.7x10-308 to 1.7x10308-1 (15 digit of precision)
Long double	10	3.4x10-4932 to 1.1x104932-1 (19 digit of precision)

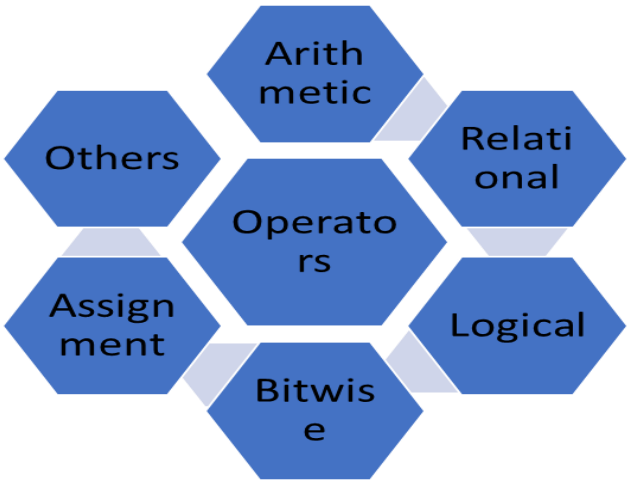
C++ Variable

Variables in C++ is a name given to a memory location. It is the basic unit of storage in a program. The value stored in a variable can be changed during program execution. A variable is only a name given to a memory location; all the operations done on the variable effects that memory location. In C++, all the variables must be declared before use.



C++ Operator

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.



Arithmetic Operator

Operator	Description	Example A=10, B=20
+	Adds two operands	**A + B will give 30**
-	Subtracts second operand from the first	**A - B will give -10**
*	Multiplies both operands	**A * B will give 200**
/	Divides numerator by de-numerator	**B / A will give 2**
%	Modulus Operator and remainder of after an integer division	**B % A will give 0** **A % B will give 10**
++	increases integer value by one	**A++ will give 11**
--	decreases integer value by one	**A-- will give 9**

Relational Operator

Operator	Description	Example A=10, B=20
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	**(A == B) is not true**
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	**(A != B) is true**
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	**(A > B) is not true**
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	**(A < B) is true**
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	**(A >= B) is not true**
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	**(A <= B) is true**

Logical Operator

Operator	Description	Example A=1, B=0
&&	Called **Logical AND** operator. If both the operands are non-zero, then condition becomes true.	**(A && B) is false**
||	Called **Logical OR** Operator. If any of the two operands is non-zero, then condition becomes true.	**(A || B) is true**

!	Called **Logical NOT** Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	**!(A && B) is true**

Bitwise Operator

Operator	Description	Example A=60, B=13
&	Binary **AND** Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary **OR** Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary **XOR** Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary **Ones Complement** Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary **Left Shift Operator**. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary **Right Shift Operator**. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

Assignment Operator

Operator	Description	Example
=	Simple assignment operator, assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, it adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, it subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, it multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, it divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A

%=	Modulus AND assignment operator, it takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	Bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	Bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Sizeof() Operator:

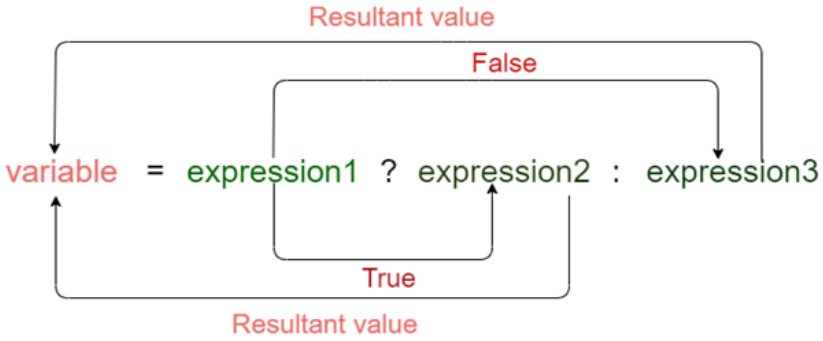
Sizeof() operator: returns the size of a variable. For example, sizeof(a), where 'a' is integer, and will return 4.

Conditional Operator: ? X : Y

Also known as ternary operator.

Syntax: Expression1? expression2: expression3;

(age>=18)? (printf("eligible for voting")) : (printf("not eligible for voting"));



C++ Program Structure

- Documentation
- Link Section
- Definition Section
- Global Declaration Section
- Function Definition Section
- Main Function

Documentation Section:

This section comes first and is used to document the logic of the program that the programmer going to code. It can be also used to write for purpose of the program. Whatever written in the documentation section is the **comment** and is **not compiled by the compiler.** Documentation Section is **optional** since the program can execute without them.

Linking Section:

The linking section contains two parts:

Header Files: Generally, a program includes various programming elements like built-in functions, classes,

keywords, constants, operators, etc. that are already defined in the standard C++ library. In order to use such pre-defined elements in a program, an appropriate header must be included in the program.
Namespaces: A namespace permits grouping of various entities like classes, objects, functions, and various C++ tokens, etc. under a single name. Any user can create separate namespaces of its own and can use them in any other program.

Definition Section:
It is used to declare some constants and assign them some value. In this section, user can define their own datatype using primitive data types.
typedef int INTEGER;
This statement tells the compiler that whenever you will encounter INTEGER replace it by int and as you have declared INTEGER as datatype you cannot use it as an identifier.

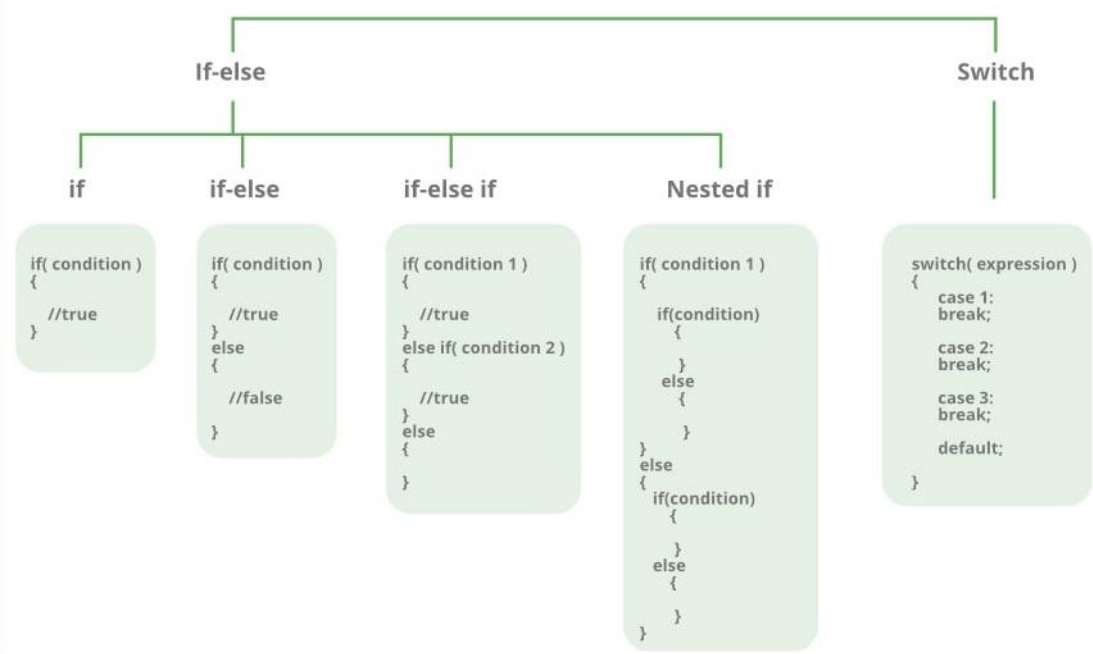
Global Declaration Section:
Here, the variables and the class definitions which are going to be used in the program are declared to make them global. The scope of the variable declared in this section lasts until the entire program terminates. These variables are accessible within the user-defined functions also.

Function Declaration Section:
It contains all the functions which our main functions need. Usually, this section contains the User-defined functions. This part of the program can be written after the main function but for this, write the function prototype in this section for the function which for you are going to write code after the main function.

Main Function:
The main function tells the compiler where to start the execution of the program. The execution of the program starts with the main function. All the statements that are to be executed are written in the main function. The compiler executes all the instructions which are written in the curly braces {} which encloses the body of the main function. Once all instructions from the main function are executed, control comes out of the main function and the program terminates and no further execution occur.

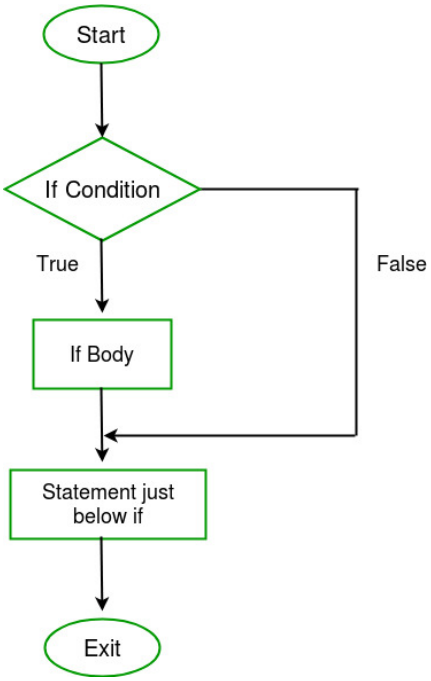
C++ Expression
C++ expressions consists of operators, constants, and variables which are arranged according to the rules of the language. It can also contain function calls which return values. An expression can consist of one or more operands, zero or more operators to compute a value. Every expression produces some value which is assigned to the variable with the help of an assignment operator.

Control Structure
The Decision making statements are used to evaluate the one or more conditions and make the decision whether to execute set of statement or not.



if statement: most simple decision-making statement.
Used to decide whether a certain statement or block of statements will be executed or not.

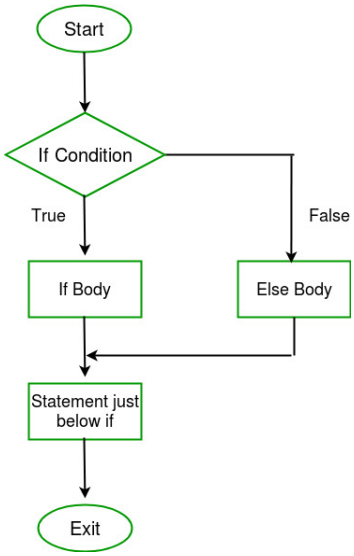
```
if(condition)
{
  // Statements to execute if
  // condition is true
}
```



If-else statement: We can use the else statement with the if statement to execute a block of code when the condition is false.

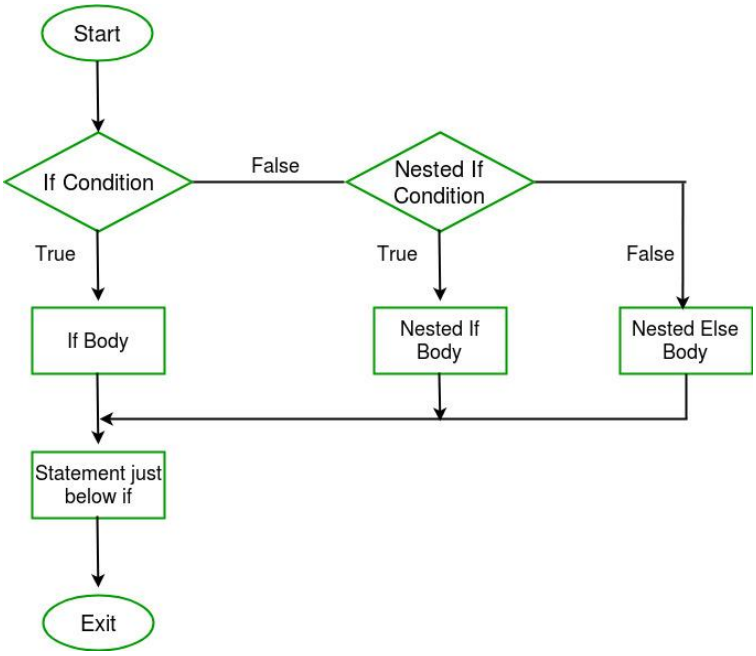
```
if (condition)
{
   // Executes this block if
```

```
   // condition is true
}
else
{
   // Executes this block if
   // condition is false
}
```



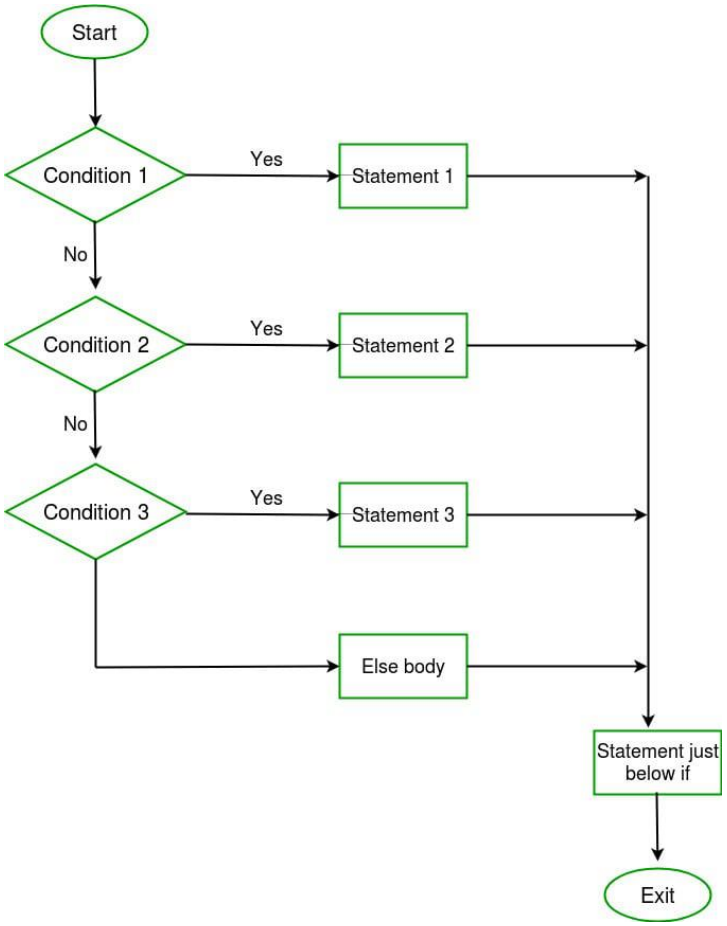
Nested If statement: Nested if statements mean an if statement inside another if statement.

```
if (condition1)
{
  // Executes when condition1 is true
  if (condition2)
  {
    // Executes when condition2 is true
  }
}
```



If-else-if ladder statement

- if (condition)
- statement;
- else if (condition)
- statement;
- .
- .
- else
- statement;



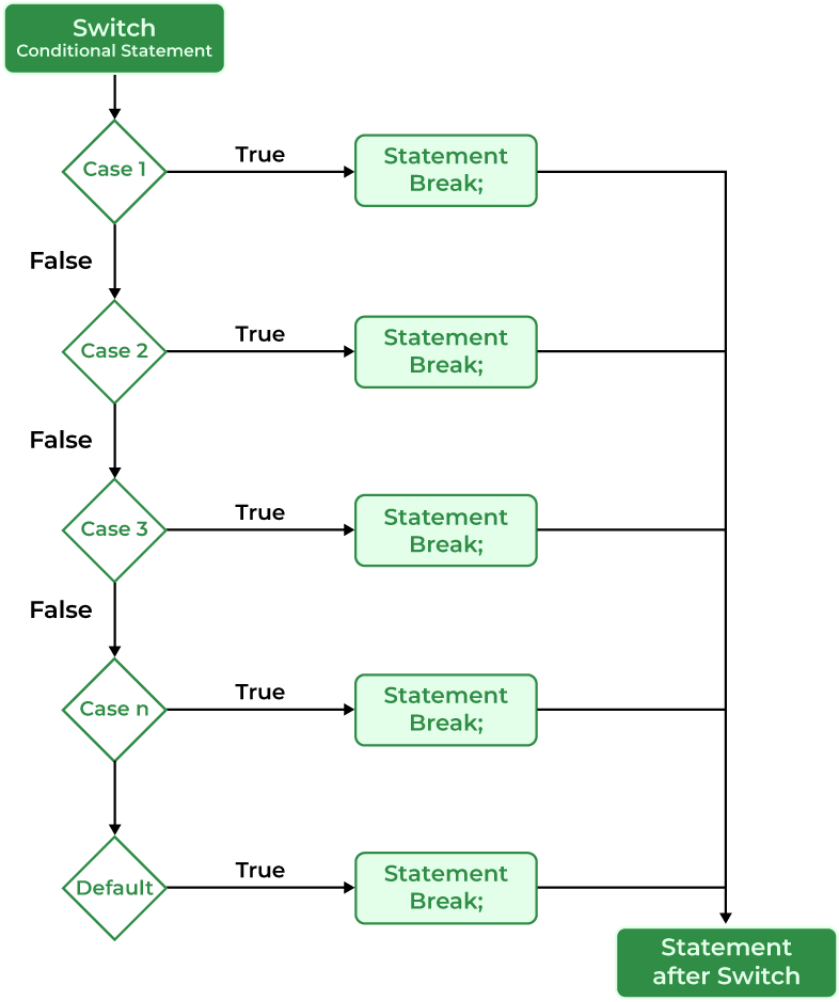
Switch Statement:

The C++ Switch case statement evaluates a given expression and based on the evaluated value, it executes the statements associated with it.

```
switch (expression) {
   case value_1:
      // statements_1;
      break;
   case value_2:
      // statements_2;
      break;
   default:
      // default_statements;
      break;
}
```

There are some rules that need to follow when using switch statements in C++:

- The case value must be either int or char type.
- There can be any number of cases.
- No duplicate case values are allowed.
- Each statement of the case can have a break statement. It is optional.
- The default Statement is also optional.

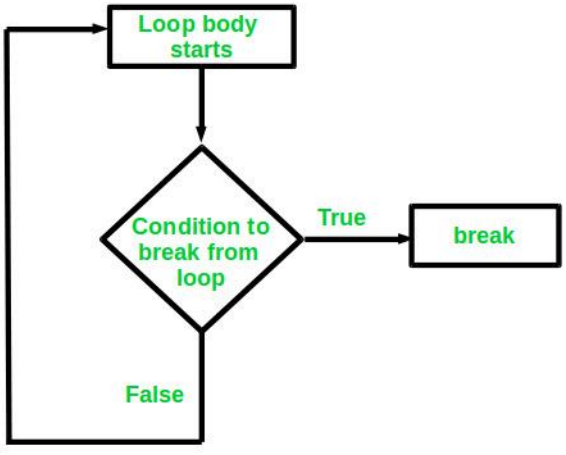


Jump Statement:

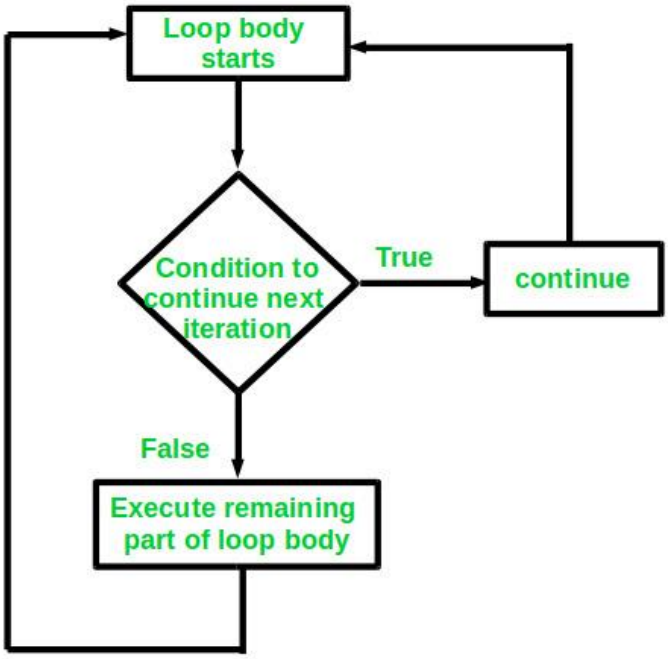
These statements are used in C or C++ for the unconditional flow of control throughout the functions in a program. They support four types of jump statements: **break, continue, goto, return**

Break statement: This loop control statement is used to terminate the loop. As soon as the break statement is encountered from within a loop, the loop iterations stop there, and control returns from the loop immediately to the first statement after the loop.

Basically, break statements are used in situations when we are not sure about the actual number of iterations for the loop or we want to terminate the loop based on some condition.



Continue Statement: The continue statement is opposite to that of the break statement, instead of terminating the loop, it forces to execute the next iteration of the loop.



Goto statement: The goto statement is referred to as the unconditional jump statement can be used to jump from one point to another within a function.
In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here, a label is a user-defined identifier that indicates the target statement. The statement immediately followed after 'label:' is the destination statement.

Return statement: The return statement returns the flow of the execution to the function from where it is called. This statement does not mandatorily need any conditional statements. As soon as the statement is executed, the flow of the program stops immediately and returns the control from where it was called. The return statement may or may not return anything for a void function, but for a non-void function, a return value must be returned.

Array
An array is a fixed-size collection of homogenous data items stored in contiguous memory locations.
It can be used to store the collection of primitive data types such as int, char, float, etc., and also derived and user-defined data types such as pointers, structures, etc.

Advantages of Array
Code Optimization (less code)
Random Access
Easy to traverse data
Easy to manipulate data
Easy to sort data

Disadvantages of Array
Fixed size

Array Declaration

data_type array_name [size];

data_type array_name [size1] [size2]...[sizeN];

Array Initialization with Declaration

data_type array_name [size] = {value1, value2, ... valueN};

Array Initialization with Declaration without size

data_type array_name[] = {1,2,3,4,5};

Access Array Elements

An array element can be accessed using the array subscript operator [ ] and the index value i of the element.

array_name [index];

```
int arr[5] = { 15, 25, 35, 45, 55 };
 // accessing element at index 2 i.e 3rd element
printf("Element at arr[2]: %d\n", arr[2]);
// accessing element at index 4 i.e last element
printf("Element at arr[4]: %d\n", arr[4]);
```

Single Dimensional Array (1-D Array)

array_name [size];

Two-Dimensional Array (2-D Array)

It can be visualized in the form of rows and columns organized in a two-dimensional plane.

array_name[size1] [size2];

Three-Dimensional Array (3-D Array)

A 3D array has exactly three dimensions. It can be visualized as a collection of 2D arrays stacked on top of each other to create the third dimension.

array_name [size1] [size2] [size3];

Function:

A function is a set of statements that take inputs, do some specific computation, and produce output.

In simple terms, a function is a block of code that only runs when it is called.

Every C++ program has at least one function, which is main().

A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.

A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

Return Type: A function may return a value. The return type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return type is the keyword void.

Function Name: This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

Function Body: The function body contains a collection of statements that define what the function does.

If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the formal parameters of the function.

The formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

While calling a function, there are two ways that arguments can be passed to a function:

Call by Value: This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

Call by Reference: This method copies the reference of an argument into the formal parameter. Inside the function, the reference is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.