CS-15
C++ and Object Oriented Programming

Unit – 1 (Part 1)

Presented By : Dhruvita Savaliya

Principles of Object Oriented Programming Tokens, and Control Statements

Presented By : Dhruvita Savaliya

- **Topics**
- Procedure – oriented programming
- Object oriented programming paradigm
- Basic concepts of object-oriented Programming
- Benefits of object-oriented programming
- Application of object-oriented programming
- What is C++?
- Application of C++
- Input/output operators
- Structure of C++ program
- Introduction of namespace
- Tokens:(keywords, identifiers,basic data types, user- defined types, derived data types, symbolic constants, type compatibility, declaration of variables, dynamic initialization of variables, reference variables)
- Operators in C++:
  - scope resolution operator,
  - member referencing operator,
  - memory management operator,
  - Manipulators
- Control Structure :
  - Conditional Control Structure :
    - simple if, if...else , nested if else, switch etc.
  - Looping control structure:
    - for, while , do...while

❖ What is Program ?

- A computer program is a sequence or set of instructions in a programming language for a computer to execute.
- It is one component of software, which also includes documentation and other intangible components.
- Program is a group of codes(variables, keywords, functions etc.) which help computer to generate output for user.
- **Example:**
  - You wish to show your name on the screen.
  - You wish to calculate addition of given values.
  - You wish to create a report card.

❖ What is Programming ?

- Programming means to arrange all the parts of program in proper order for meaningful purpose.

❖ What is Language ?

- Language is a way of communication.
- Using any language we can send or receive information or command.

❖ What is Programming Language ?

- A programming language is **a system of notation for writing computer programs.**
- Programming Language is a platform where we can arrange particular command list to perform such task in the form of program.
- **Example :**
- C , C++ , C#.net , VB.net etc…

❖ Types of Programming Language :

- In human concept, there are two types of languages like:
  - **Symbolic Language**
  - **Words Language**
- like, English, Gujarati, Hindi etc.

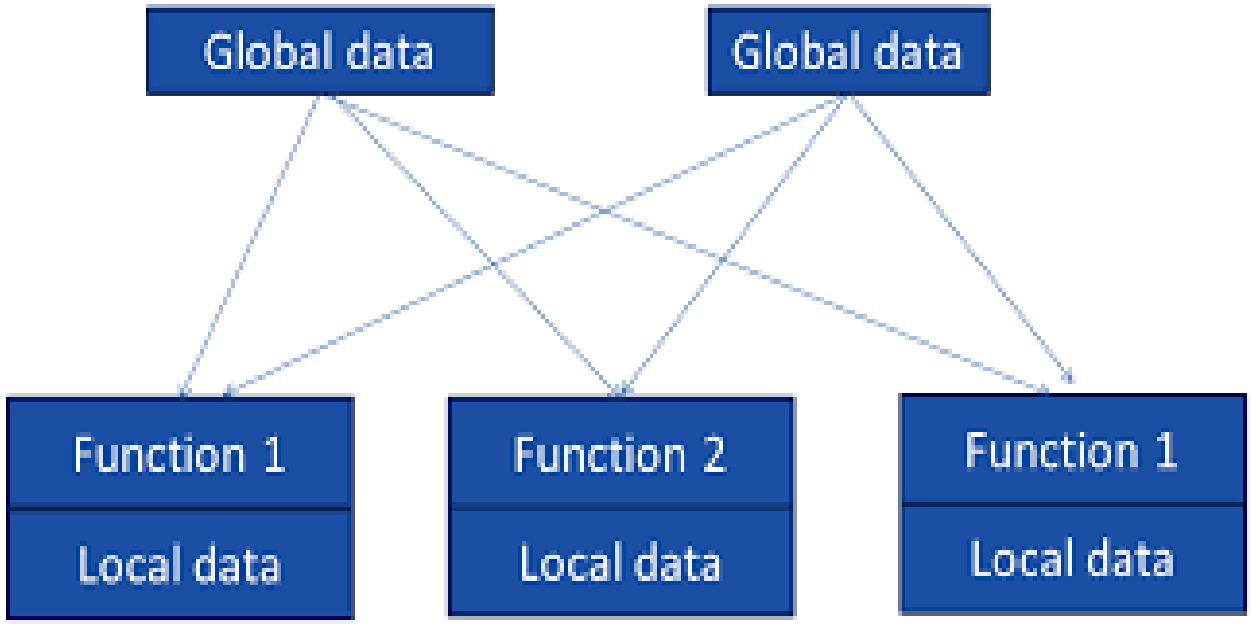
- Same as that there are two types of programming languages available in Computer :
  - **POP (Procedure Oriented Programming)**
  - **OOP (Object Oriented Programming)**

❖ Procedural Oriented Programming :

- A procedural language is a sort of computer programming language that has a set of functions, instructions, and statements that must be executed in a certain order to accomplish a job or program.
- In general, procedural language is used to specify the steps that the computer takes to solve a problem.

- **Example :**

- FORTRAN
- ALGOL
- BASIC
- COBOL
- Pascal
- C



- **Characteristics of POP :**
  - Emphasis is on doing things (algorithms).
  - Large programs are divided into smaller programs known as functions.
  - Most of the functions share global data.
  - Data move openly around the system from function to function.
  - Functions transform data from one form to another.
  - Employs ***top-down* approach** in program design.
- POP basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as *functions*.
- Serious drawback with the procedural approach is that it does not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the elements of the problem.

Advantage of Procedural Languages :

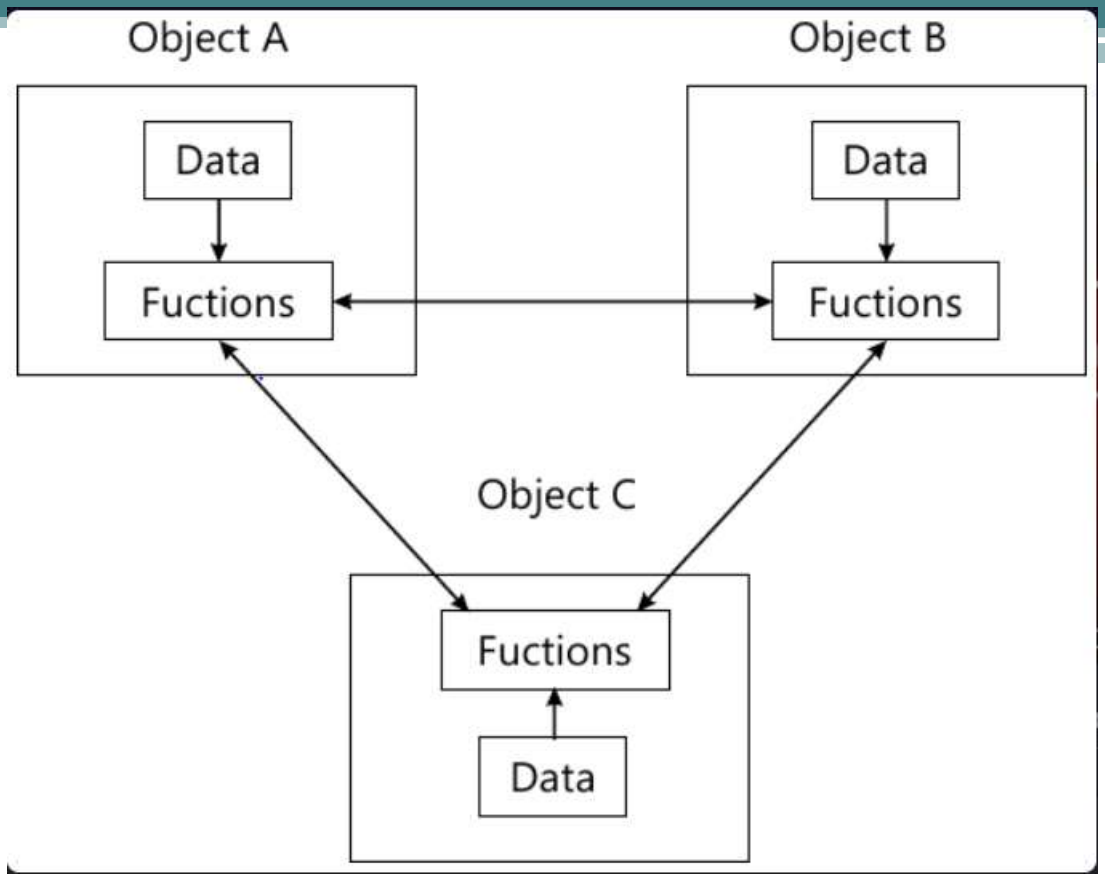
- The program of Procedural Programming language forwardness apace with the utilization of interpreters and accumulators.
- This language clarifies the source code and can be understood easily.
- Without the need for copying this, the code may reuse in various piece of code.
- For various purposes, the Procedural Programming language utilizes different parts of memory.
- There are multiple General-Purpose programming languages, which support it.
- It makes it easy to track as it flows the program in a linear direction.

Disadvantages of Procedural Languages :

- When Procedural languages are employed, its program is harder to write.
- Also, with a real-world problem, this language is not very practical.
- It builds less safety inviting because the information is available to the whole code.
- It can form a complex program as it has the potential to solve real-world problems.
- The information is vulnerable in this, and it is not practicable with a true issue.
- **Uses of Procedural languages :**
- Procedural programming languages are utilized by content and programming developers. For creating programs and showing the ideal yield, they use factors, restrictive proclamations, and capacities.

❖object oriented programming paradigm:

- **Alan Kay** coined the term “Object Oriented Programming”.
- It was created between 1961 and 1962 and published in his Sketchpad Thesis in 1963.
- Object-oriented programming – As the name suggests 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.



- **Characteristics of OOP :**
- In OOP the importance is on **data in place of function.**
- Large programs are divided into **objects.**
- The data of objects is hidden and can't be accessed by outside functions.
- Objects can pass information through functions.
- It follows **Bottom-Up approach**.

❖ Top-Down Approach :

- The Top-Down approach means the programmer stats the program with main function and then thinks of other steps to take such as functions and other coding.
- So the program execution also starts from top because of main function and goes down eventually is as follows.
- **Top-Down Approach** is an approach to design algorithms in which a bigger problem is broken down into smaller parts.
- 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.
- Also, there is 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.
- Top-down approach is generally used with documentation of module and debugging code.

Advantages of top-down management

Disadvantages of top-down management

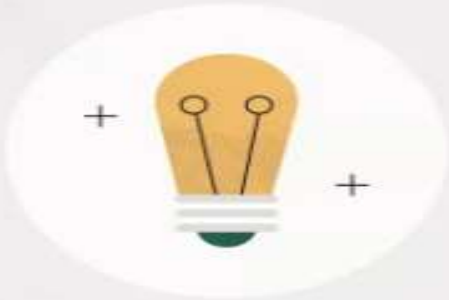
❖Bottom-Up Approach :

- **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. Therefore, it uses composition approach.
- In Bottom-Up approach the programmer has to think for basic functionalities needed by program and then develop them.
- So after developing the functions and other structures the main function comes in last where all the functionalities will be used there.
- So the execution starts from bottom because the main function and goes upside to call other functions which are at top.
- object oriented programming paradigm such as **C++**, **Java**, **Python** etc...
- Data encapsulation and data hiding is also implemented in this approach.
- The bottom-up approach is generally used in testing modules.

Advantages of bottom-up management

More informed decisions

Better team morale

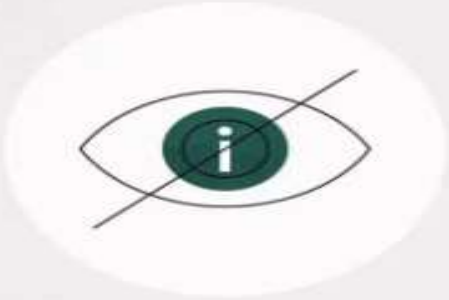


More room for creativity

Disadvantages of bottom-up management

Low momentum

Shift in leadership dynamics



Lack of high-level insight

❖Difference between Top-Down Approach V/S Bottom-Up Approach

TOP DOWN APPROACH	BOTTOM UP APPROACH
In this approach We focus on breaking up the problem into smaller parts.	In bottom up approach, we solve smaller problems and integrate it as whole and complete the solution.
Mainly used by structured programming language such as COBOL, Fortran, C, etc.	Mainly used by object oriented programming language such as C++, C#, Python.
Each part is programmed separately therefore contain redundancy.	Redundancy is minimized by using data encapsulation and data hiding.
In this the communications is less among modules.	In this module must have communication.
It is used in debugging, module documentation, etc.	It is basically used in testing.
In top down approach, decomposition takes place.	In bottom up approach composition takes place.

TOP DOWN APPROACH	BOTTOM UP APPROACH
In this top function of system might be hard to identify.	In this sometimes we can not build a program from the piece we have started.
In this implementation details may differ.	This is not natural for people to assemble.
Pros- •Easier isolation of interface errors •It benefits in the case error occurs towards the top of the program. •Defects in design get detected early and can be corrected as an early working module of the program is available.	Pros- •Easy to create test conditions •Test results are easy to observe •It is suited if defects occur at the bottom of the program.
Cons- •Difficulty in observing the output of test case. •Stub writing is quite crucial as it leads to setting of output parameters. •When stubs are located far from the top level module, choosing test cases and designing stubs become more challenging.	Cons- •There is no representation of the working model once several modules have been constructed. •There is no existence of the program as an entity without the addition of the last module. •From a partially integrated system, test engineers cannot observe system-level functions. It can be possible only with the installation of the top-level test driver.

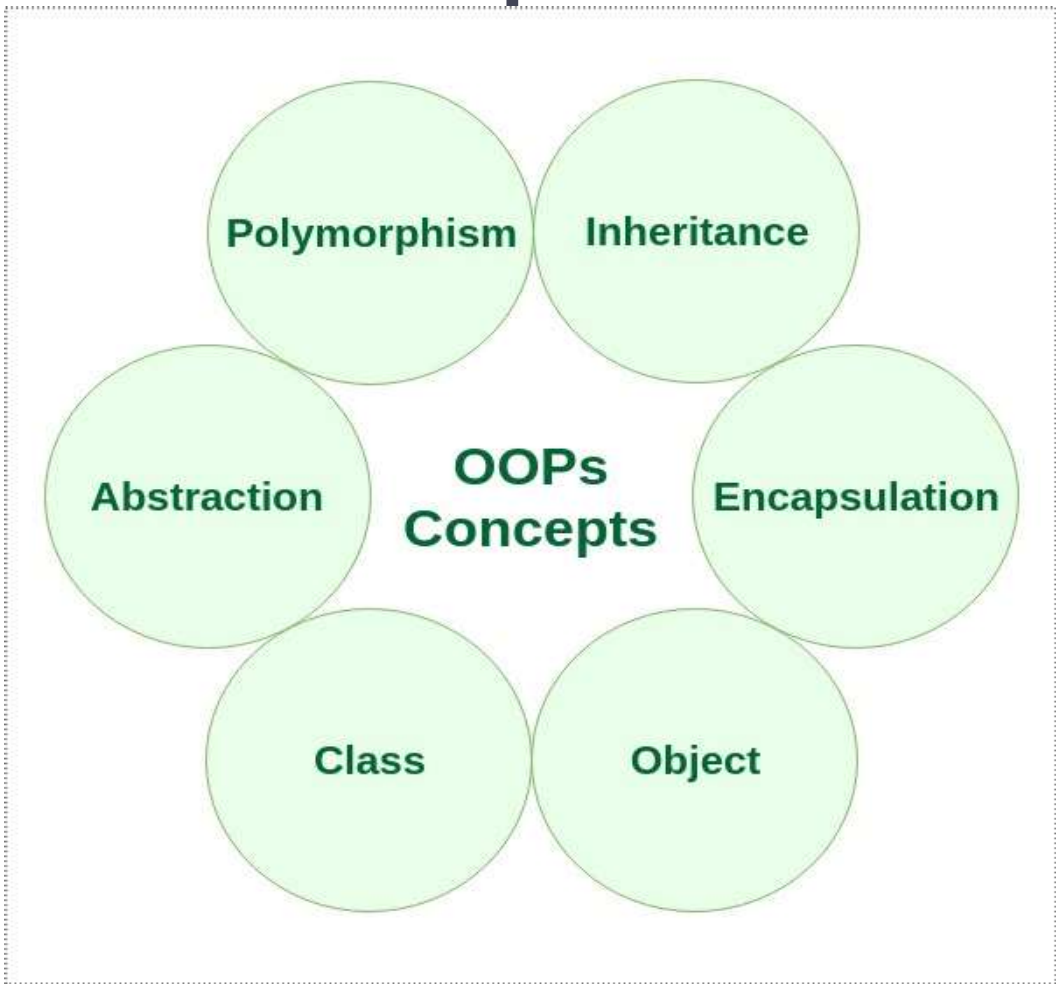
❖Difference between OOP V/S POP

Parameter	OOP	POP
Definition	This type of programming language uses objects and classes for creating models.	This programming language uses a step-by-step approach for breaking down a task into a collection of routines and variables by following a sequence of instructions.
Full Name	Object-Oriented Programming	Procedure Oriented Programming
Approach	Bottom-up approach	Top-down approach
Polymorphism	Method overloading and overriding are used in OOP to achieve polymorphism.	It doesn't support polymorphism.
Inheritance	Supports	Do not support
CodeReusability	Supports	Don't support
Security	Data handling is possible in OOP due to programming.	It is less secure than OOP.
Problem-Solving	Used for solving big problems.	Not suitable for big problems.
Example	C++, JAVA, C#, .NET	C, FORTRAN

Basic concepts of OOP / Features of OOP How many concept of OOP? And what is the meaning of all the concept :

1. Class
2. Objects
3. Encapsulation
4. Abstraction
5. Polymorphism
6. Inheritance

- Dynamic Binding
- Message Passing



Class :

- A **class** is a data-type that has its own members i.e. data members and member functions.
- It is the blueprint for an object in object oriented programming language.
- It is the basic building block of object oriented programming in C++. The members of a class are accessed in programming language by creating an instance of the class.
- **Some important properties of class are –**
- **Class** is a user-defined data-type.
- A class contains members like data members and member functions.
- **Data members** are variables of the class.
- **Member functions** are the methods that are used to manipulate data members.
- Data members define the properties of the class whereas the member functions define the behaviour of the class.

- A class can have multiple objects which have properties and behaviour that in common for all of them.
- **Syntax :**

```
class class_name
{
    data_type data_name;
    return_type method_name(parameters);
}
```

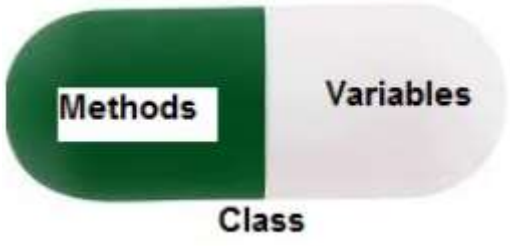
Object :

- An **object** is an instance of a class.
- It is an entity with characteristics and behaviour that are used in the object oriented programming.
- An object is the entity that is created to allocate memory.
- A class when defined does not have memory chunk itself which will be allocated as soon as objects are created.
- **Syntax :**

  class_name object_name;

Encapsulation :

- **Encapsulation** in object oriented programming **encapsulation** is the concept of wrapping together of data and information in a single unit.
- A formal definition of encapsulation would be: encapsulation is binding together the data and related function that can manipulate the data.
- Using the method of encapsulation, the programmer cannot directly access the data.
- Data is only accessible through the object of the class.
  - The data member is not accessible to the outside world, and only those functions which are wrapped in the (cover) enclose class can access it.
  - Declaring private members and making then unaccessible from outside the class is another feature of oop known as data hiding.



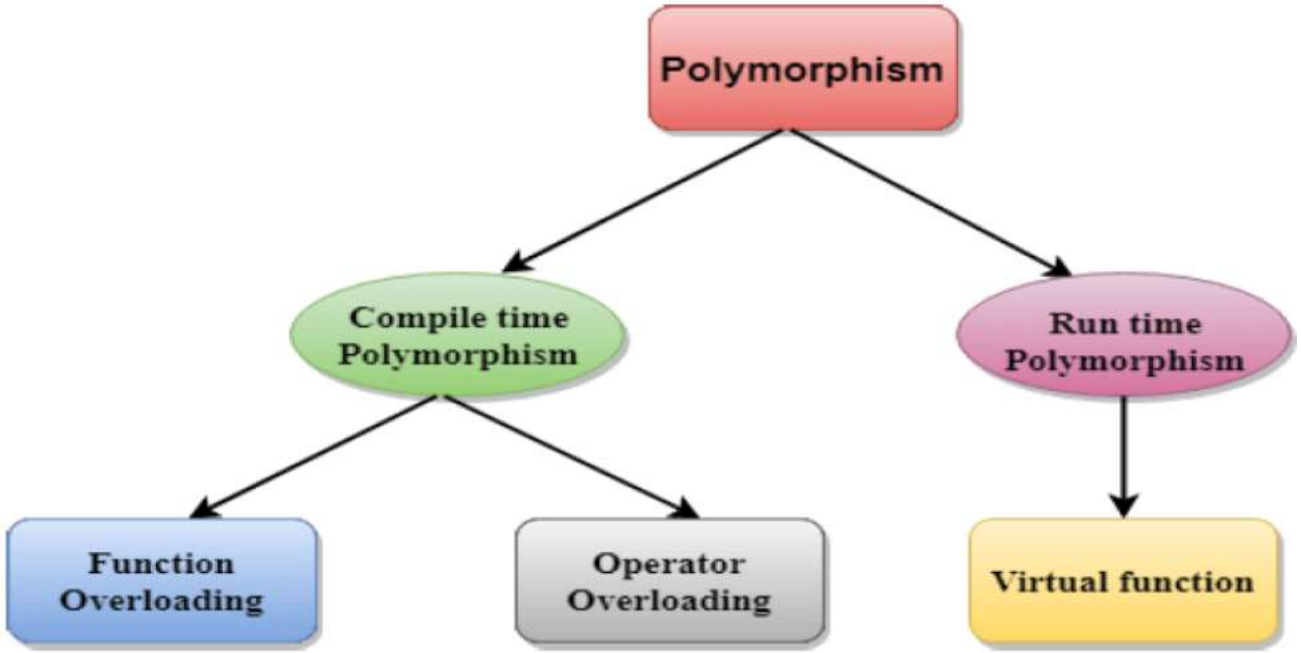
Abstraction :

- Data abstraction is also known as data hiding and showing only relevant data to the final user.
- Creating new data type using encapsulated items, that are well suited to an application to be programmed, is known as abstraction.
- The data type created by the data abstraction process are known as abstract data types (ADT).
- Data abstraction having the properties of built in datatypes and set of permitted operators.
- In C++ programming language write two ways using which we can accomplish data abstraction –
  - using class
  - using header file

Polymorphism :

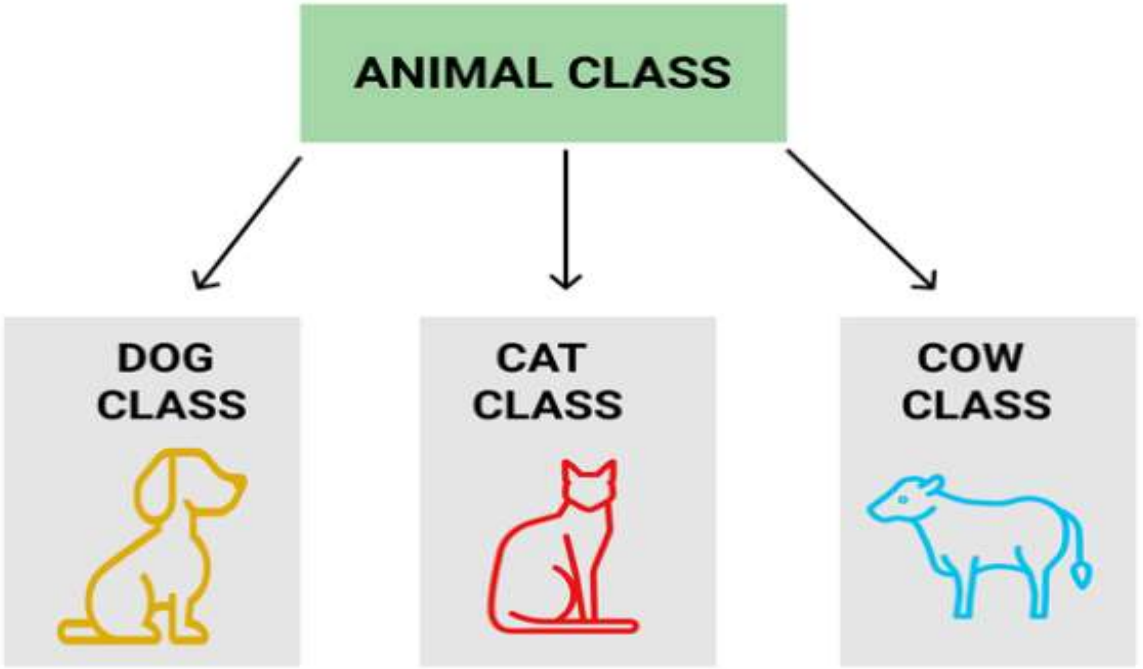
- Polymorphism is a Greek term, it will created with two different word it is poly + morph = polymorphism.
- Poly = many and morph = form's.
- The word polymorphism means having More than one form in single entity.
- In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
- A person at the same time can have different characteristics.
  - A man at the same time is a father, a husband, and an employee. So the same person possesses different behavior in different situations.
- This is called polymorphism. An operation may exhibit different behaviors in different instances.
- The behavior depends upon the types of data used in the operation.
  - + operator. It perform addition if two numeric values passed (2+3=5).
  - It perform concatenation if to string passed ("abc" + "xyz" = "abcxyz").

- We can implement polymorphism in C++ using the following ways:
  - Function overloading
  - Operator overloading
  - Function overriding
  - Virtual functions
- ***Operator Overloading***: The process of making an operator exhibit different behaviors in different instances is known as operator overloading.
- ***Function Overloading***: Function overloading is using a single function name to perform different types of tasks. Polymorphism is extensively used in implementing inheritance.



Inheritance :

- **Inheritance** it is the capability of a class to inherit or derive properties or characteristics other class.
- Inheritance is very important and object oriented program as it allows reusability i.e. using a method defined in another class by using inheritance.
- The class that derives properties from other class is known as child class or subclass and the class from which the properties are inherited is base class or parent class.
  - **Super Class**: The class whose properties are inherited by a sub-class is called Base Class or Super class or parent class.
  - **Sub Class**: The class that inherits properties from another class is called Sub class or Derived Class or child class.
  - **Reusability**: Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.



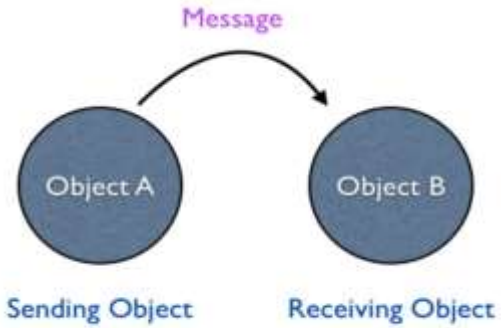
- C++ programming language supports the following types of inheritance -
  - single inheritance
  - multiple inheritance
  - multi level inheritance
  - Hierarchical inheritance
  - hybrid inheritance

Dynamic Binding :

- In dynamic binding, the code to be executed in response to the function call is decided at runtime.
- C++ has virtual functions to support this. Because dynamic binding is flexible, it avoids the drawbacks of static binding, which connected the function call and definition at build time.

Message Passing :

- Objects communicate with one another by sending and receiving information.
- A message for an object is a request for the execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results.
- Message passing involves specifying the name of the object, the name of the function, and the information to be sent.



Message Passing

❖Benefits of object-oriented programming:

1. Easier Troubleshooting
2. Code Reusability
3. Data Redundancy
4. Code Flexibility
5. Polymorphism Flexibility
6. Better Productivity
7. Security
8. Enhanced Problem-Solving
9. Optimizes Software Design
10. Maintenance of Code

Application of object-oriented programming :

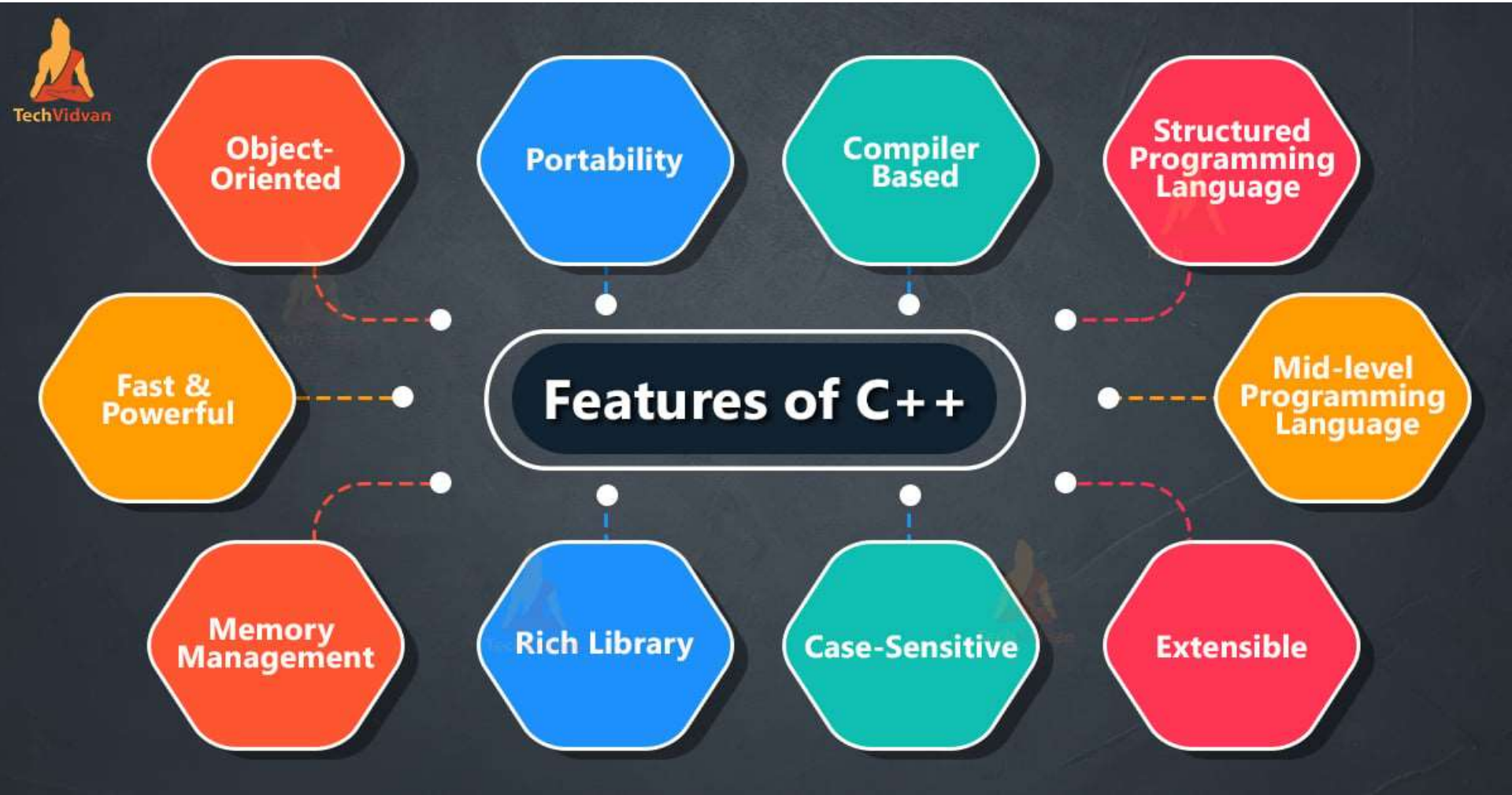
- Popularity of OOPs in the development of most software systems with ease, has created a great deal of excitement and interest among software communities.
- OOP finds its application from design of database systems to the future generation operating systems, which have computing, communication, and imaging capabilities built into it.
- Today, OOP is used extensively in the design of Graphics User Interfaces on systems such as windows.

- **The promising areas for application of oop include:**
- 1. Client-Server Systems
- 2. Object-Oriented Databases
- 3. Real-Time System Design
- 4. Simulation And Modelling System
- 5. AI Expert Systems
- 6. Hypertext, hypermedia and expertext
- 7. Neural networks and parallel programming
- 8. CAM/CAD systems. (CAM=computer-aided manufacturing / CAD=computer-aided design)

❖ What is C++ ?

- Basically C++ is not a new language but it is just and updation to C programming language.
- C++ is an object oriented programming language.
- C++ is **developed by Bjarne Stroustrup at AT&T Bell Laboratories in New Jersey, USA in the early 1980's.**
- Earlier the name of the language was "C with classes" but after sometimes it was renamed to C++.
- ++ is an increment operator of C language.
- Here,C++ is an increment to the C language.

Features of C++ :



❖ Application of C++ :

1. Operating Systems :

C++ was used in developing most of the operating systems in the world, from Microsoft Windows to macOS to Linux.

2. GUI Based Applications :

C++ was also used in developing most of the GUI (Graphical User Interface) applications that we use regularly.

For example, Photoshop, Illustrator, Adobe Premier.

3. Browsers :

Since C++ has faster execution, it helps in developing rendering engines in browsers.

C++ supported all the major browsers like Google Chrome, Mozilla (Firefox and Thunderbird), Safari, Opera, and many more.

4. Graphics :

C++ has high performance and speed which comes in handy for making applications that require 3D animation, modeling, image processing, and real-time simulations.

- **5. Database Software :**
  Along with browsers and GUI applications, C++ is also used in developing the most important DBMS engines like MySQL, Postgres, Redis, and Oracle.
- **6. Libraries :**
  C++ has got a great set of in-built standard libraries which help in developing high-level machine learning and mathematical computation libraries.

- ***Real-World Applications of C++***

1. Games
2. Graphics User Interface
3. Web Browsers
4. Advance Computations and Graphics
5. Database Software
6. Operating Systems
7. Enter prose Software

❖ Input / Output Operator :

- The C++ language does not define any statements to do input or output(IO).
- Instead, C++ includes an extensive Standard Library that provides IO .
- For the purpose of input & output we will make use of the iostream(Input Output Stream) Library.
    - The iostream library is made up of two types :

      istream and ostream

      Input Stream Output Stream
- **What is Stream ?**
- A Stream is Sequence of characters read from or written to an IO device.
- **Standard Input & Output Objects :**
- **cin** (Console Input) → Standard Input
- **cout** (Console output) →Standard Output

- **Output Operator :- 'cout' object & '<<' operator or Standard Output Stream cout :**
- This object and operator is used to display information on the monitor.
- **Syntax :**
- cout << data item ;
- where, 'cout' is a pre-defined object which represents standard output stream towards monitor / screen
- '<<' is known as 'insertion operator' or 'put to operator' or 'output operator'.
- It towards the object & cout.
- data item is any constant, variable or expression.
- **Example :**

```
#include<iostream>
void main ( )
{
        int a;
        cout<<"hello BCA"<<a;
}
```

- We can use more than one insertion operators with a single object 'cout' to display multiple data items. This is referred cascaded output operation.

```
cout << a << b << c;
cout << a
<< b
<<c;
```

- **Input Operator :- 'cin' object & '>>' operator or Standard Input Operator :**
- This object and operator is used to enter value of variable from keyboard.
- **Syntax :**

  cin >> variable ;
- Where, 'cin' is a pre-defined object which represents standard input stream from keyboard.
- '>>' is known as 'extraction operator' or 'get from operator' or 'input operator'.
- It executes the value from input stream & directs it towards variable on its right.
- variable is any valid variable name.
- **Example :**

```
#include<iostream>
void main ()
{
        int a;
        cin>>a;
        cout<<a;
}
```

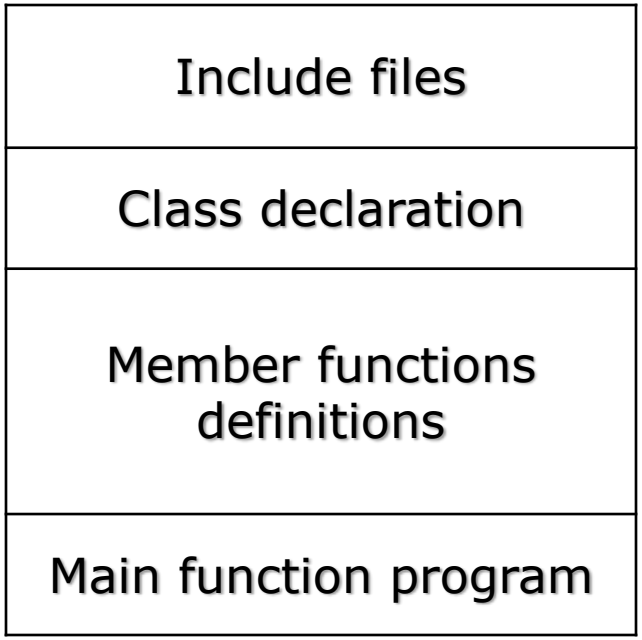
- We can use multiple extraction operators with the single object 'cin' to enter multiple values for multiple variables.

```
cin >> a >> b >> c;
cin >> a
    >> b
    >> c ;
```

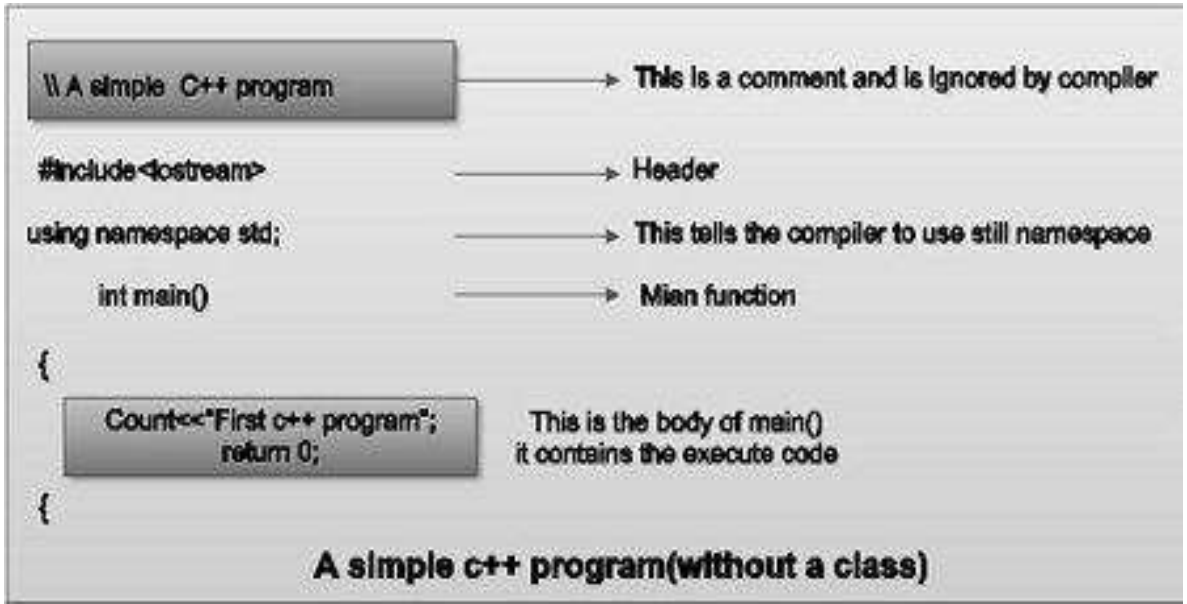
❖ C++ Program Structure :

1. Documentation.
2. File Include Section.
3. Global Variable Declaration.
4. Class Declaration.
5. Function Definition Section.
6. Main Function.

- Programs are a sequence of instructions or statements.
- These statements form the structure of a C++ program.
- C++ program structure is divided into various sections, namely, headers, class definition, member functions definitions and main function.



- Note that C++ provides the flexibility of writing a program with or without a class and its member functions definitions.
- A simple C++ program (without a class) includes comments, headers, namespace, main() and input/output statements.
- Comments are a vital element of a program that is used to increase the readability of a program and to describe its functioning.
- Comments are not executable statements and hence, do not increase the size of a file.



❖ Namespace :

- To understand the concept of namespace, consider a situation in which you need to create to classes with same name.
- In any operating system we cannot create two files with same name in same folder.
- If we have to create two files with same name then we have to create those files in different folders.
- The namespace is the same concept to the folder.
- Namespace is used to resolve the name clash problem in your program.
- Namespace is just a collection of classes.
- We can add our classes, functions, variables and related code in a namespace to create a useful container of related code.

- Now we can create another namespace and add classes, functions or variables with same name of the old namespace without any clash.
- To create a namespace we have to use **namespace keyword**
- **Syntax :**

```
namespace namespace_name
{
        // required statements...
}
```

- To use classes, functions or variable from name space we have to include a statement saying that you are using that namespace in your program.

  using namespace **namespace_name**

- **NOTE :**
- The namespace concept will not run under Turbo C++ compiler.
- We can use namespace in advanced C++ editor such as Microsoft Visual C++.

❖ Tokens :

- ***A C++ program is composed of tokens which are the smallest individual unit.*** Tokens can be one of several things, including keywords, identifiers, constants, operators.

❖ Keyword :

- ***Keywords in C++ refer to the pre-existing, reserved words, each holding its own position and power and has a specific function associated with it.***
- It is important to note that we cannot use C++ keywords for assigning variable names as it would suggest a totally different meaning entirely and would be incorrect.

alignas	alignof	asm	auto	bool	break
case	catch	char	char16_t	char32_t	class
const	constexpr	const_cast	continue	decltype	default
delete	double	do	dynamic_cast	else	enum
explicit	export	extern	FALSE	float	for
friend	goto	if	inline	int	long
mutable	namespace	new	noexcept	nullptr	operator
private	protected	public	register	reinterpret_cast	return
short	signed	sizeof	static	static_assert	static_cast
struct	switch	template	this	thread_local	throw
TRUE	try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile	wchar_t
while	–	–	–	–	–

Identifiers :

- C++ allows the programmer to assign names of his own choice to variables, arrays, functions, structures, classes, and various other data structures called identifiers.
- The programmer may use the mixture of different types of character sets available in C++ to name an identifier.
- **Rules for C++ Identifiers**

1. **First character:** The first character of the identifier in C++ should positively begin with either an alphabet or an underscore. It means that it strictly cannot begin with a number.
2. **No special characters:** C++ does not encourage the use of special characters while naming an identifier. It is evident that we cannot use special characters like the e*xclamatory mark* or the *"@"* symbol.
3. **No keywords:** Using keywords as identifiers in C++ is strictly forbidden, as they are reserved words that hold a special meaning to the C++ compiler. If used purposely, you would get a compilation error.
4. **No white spaces:** Leaving a gap between identifiers is discouraged. White spaces incorporate blank spaces, newline, carriage return, and horizontal tab.
5. **Word limit:** The use of an arbitrarily long sequence of identifier names is restrained. The name of the identifier must not exceed 31 characters, otherwise, it would be insignificant.
6. **Case sensitive:** In C++, uppercase and lowercase characters connote different meanings.

❖ Constants :

- As the name suggests, constant are fixed values that should not be chang1ed throughout the program execution.
- Constants can be integer, floating point, characters or strings.
- **Example :**

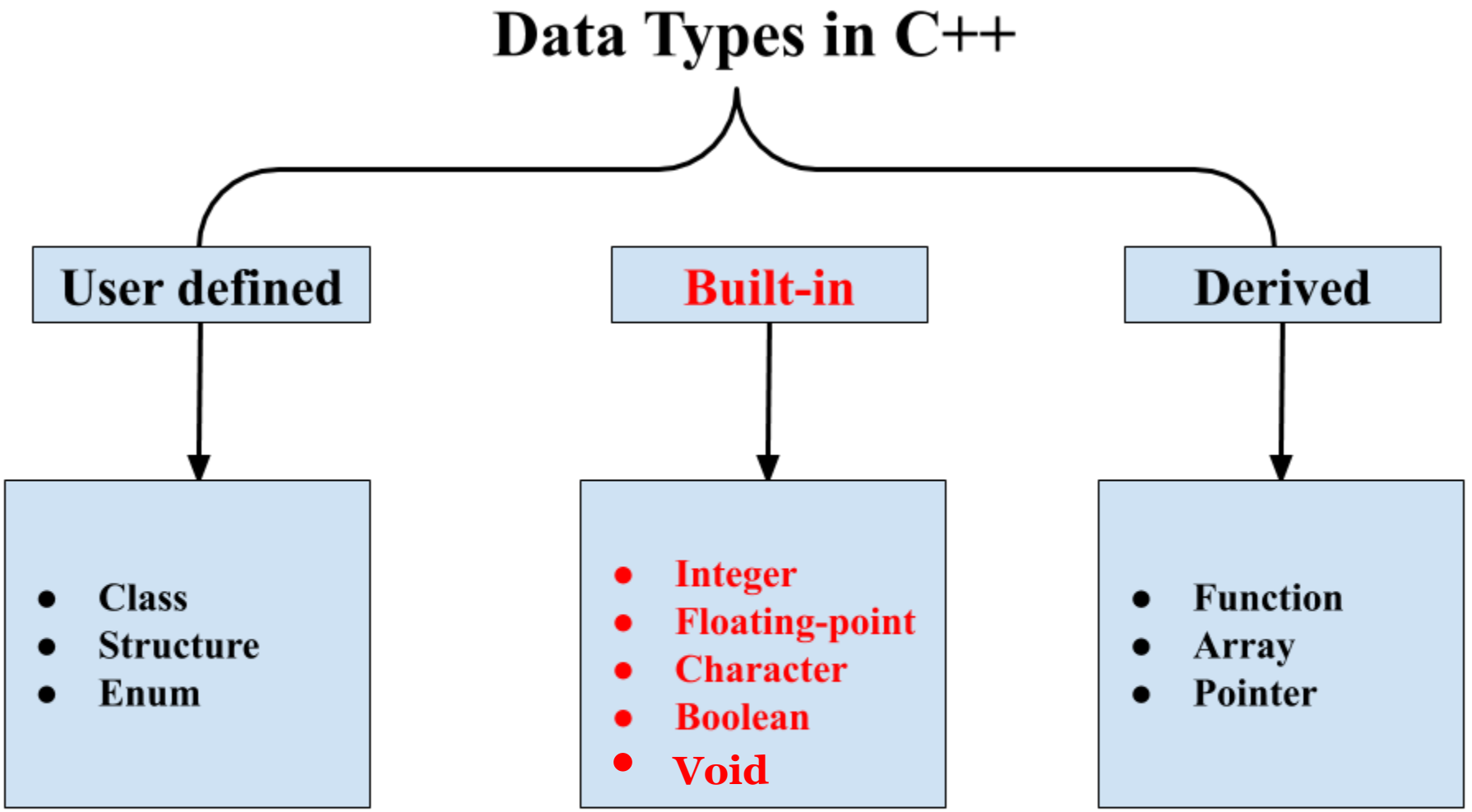
1. Integer Constant: 123, 555, 2569 etc
2. Floating Point Const: 1.23, 40.10, 1.75 etc
3. Character Constant: 'a', 'b', 'x', 'Y' etc
4. String Constant: 'ABC', 'Computer' etc

❖ Strings :

- Just like characters, ***strings in C++*** are used to store letters and digits. Strings can be referred to as an array of characters as well as an individual data type.
- It is enclosed within double quotes, unlike characters which are stored within single quotes. The termination of a string in C++ is represented by the null character, that is, **'\0'.** The size of a string is the number of individual characters it has.
- **In C++, a string can be declared in the following ways:**
- *char name[30] = "Hello!"; // The* compiler *reserves 30 bytes of memory for the string.*
- *char name[] = "Hello!"; // The compiler reserves the required amount of memory for the string.*
- *char name[30] = { 'H' , 'e' , 'l' , 'l' , 'o'};; // This is how a string is represented as a set of characters.*
- *string name = "Hello" // The compiler reserves 32 bytes of memory.*

❖ Special Symbols :

- Apart from letters and digits, there are some special characters in C++ which help you manipulate or perform data operations.
- Each special symbol has a specific meaning to the C++ compiler.
- **[ ] { } ( ) , # * ~ .**


Data Types in C++
User defined
Built-in
Derived
Class
Structure
Enum
Integer
Floating-point
Character
Boolean
Void
Function
Array
Pointer

❖ Basic Data Type / Built-in Data Type :

- These data types are built-in or predefined data types and can be used directly by the user to declare variables.
- example: int, char, float, bool, etc.

❑ **Integer Data type :**

- Integer data types represent whole numbers without a fractional or decimal part. They can be signed (positive, negative, or zero) or unsigned (only positive or zero).
- **Example :**

```
int signedInt = -42;
unsigned int unsignedInt = 123;
```

❑ **Character :**

- Character data types represent individual characters from a character set, like ASCII or Unicode. In C++, ‘char’ is commonly used to represent characters.
- **Example :**

```
char myChar = 'A';
```

- **Boolean :**
- Boolean data types represent binary values, typically used for true (1) or false (0) conditions. In C++, `bool` is used for Boolean data.
- **Example :**

  bool isTrue = true; bool isFalse = false;The two boolean variables "isTrue" and "isFalse" are declared in this code with the values true and false, respectively.

- **Floating Point :**
- Floating-point data types represent numbers with a fractional part. In C++, 'float' is a single-precision floating-point type.
- **Example :**

  float myFloat = 3.14159f;

- **Valueless or Void :**
- The void data type in C++ is used to indicate that a function does not return any value or to declare generic pointers that do not point to a specific data type.
- **Example :**

  void myFunction() { // This function does not return a value }

Data Type	Size (In Bytes)	Range
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
float	4	
double	8	
long double	12	

❖ User Defined Data Type :

- *In a programming language,* ***User Defined Data Types*** *are defined by the users in the program as per their needs in order to store data either of the same or different types as per the requirement.*
- ***User Defined Data types*** *or Composite Data types are derived from more than one built-in data type which is used to store complex data.*

- **Class** – A User-defined data type that holds data members and functions whose access can be specified as private, public, or protected.
  It uses the '**class**' keyword for defining the data structure.

```
class <classname>
{
        private:
                Data_members;
                Member_functions;
        public:
                Data_members;
                Member_functions;
};
```

- **Structure** – A structured data type is used to group data items of different types into a single type. For Eg. Structure can be Address, in which it further contains information such as Flat number, Building name, street, city, state, pin code, etc.
  It is defined by using the '**struct**'
  Example :

```
struct stud_id
{
        char name[20];
        int class;
        int roll_number;
        char address[30];
};
```

- **Union –** Union is a type of data structure where all the members of that union share the same memory location.

  If any changes made in the Union, it will be visible to others as well.

  The '**union**' keyword is used to define this type of User-defined data type.

  Example :

  union test

  {

  int x, y;

  };

- **Enumeration –** It helps assign names to integer constants in the program.

  The keyword '**enum**' is used.

  It is used to increase the readability of the code.

  Example :

  enum fruits{apple, mango, grapes};

  Here,applewill be assigned with **0** mango **1and grapes 2**

  We can also assign specified numer to particular item to…

- Enum week{mon=1, tue, wed, thu, fri, sat, sun};

- **Typedef –** Typedef defines a new name for an existing data type.

  It does not create a new data class.

  It makes code readability easy and gives more clarity to the user.

  Example :

  typedef int var;

  var a=10; //var is new data type of int that we create.

❖ Derived Data Type :

- The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.

  Function

  Array

  Pointer

Function :

- A **Function** is a block of code or program segment that is defined to perform a specific well-defined task.
- A function is generally defined to save the user from writing the same lines of code again and again for the same input.
- All the lines of code are put together inside a single function and this can be called anywhere required. main() is a default function that is defined in every program of C++.
- **Syntax**

  ReturnType FunctionName(parameters);

  Example :

```
void fun(int a,int b)
{
      cout << “sum is “<<a+b;
}
```

Array :

- An **Array** is a collection of items stored at continuous memory locations.
- The idea of array is to represent many instances in one variable.
- **Syntax :**
  DataType ArrayName[size_of_array];
- **Example :**

```
int arr[5];
arr[0] = 5;
arr[2] = -10;
```

Pointers :

- **Pointers** are symbolic representation of addresses.
- They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures.
- **Syntax :**
- datatype *var_name;
- **Example :**

```
int var = 20;
 int* ptr;
 ptr = &var;
 cout << "Value at ptr = " << ptr << "\n";
 cout << "Value at var = " << var << "\n";
 cout << "Value at *ptr = " << *ptr << "\n";
```

❖ Symbolic Constants :

- Symbolic constants are nothing more than a label or name that is used to represent some fixed value that never changes throughout the course of a program.
- For example, one might define PI as a constant to represent the value 3.14159.
- In C++ we can create symbolic constant by many ways :
- Using the **const keyword**

```
const int size=15;
char name[size];
```

- Using **enum keyword**

```
enum{x=100,y,z};  //value of x is 100 value of y is 101 value of z is 102
```

- Using **#define directive…**

```
#define var 100
```

- Constant cannot redefine during program execution.

Type Compatibility :

- Type compatibility refers to whether the types of all variables are compatible to one another or not when written in an expression.
- **Example :**
- 3 variable of different data types in an expression must be compatible to each other.
- Example :

```
int a=10;
float b=20;
double c;
c=a+b;
```

- Here, in expression c=a+b, a(int) and b(float) and c(double) all are different but it executes program without error because of compatibility.

❖ Declaration of Variables :

- Variables are used to store values.
- variable name is the name of memory location where value is stored.
- It must be alphanumeric, only underscore is allowed in a variable name.
- It is composed of letters, digits and only underscore. It must begin with alphabet or underscore.
- It can not be begin with numeric.
- Declaration will allocate memory for specified variable with garbage value.
- **Syntax :**

  Data-Type Variable-name;
- **Example :**

  int a;
  float b;
  char c;

❖ Dynamic Initialization of Variable :

- As the declaration of variables, all the variables must be initialized before using them.
- In C++ we can initialize your variables dynamically (runtime) also.
- So the variable will get its initial value when you run the program.
- **Example :**

```
void main()
{
        int number;
        cout<<"Enter Number :";
        cin>>number;
        double range=number+1;//Dynamic initialization
        for(int x=1;x<=number;x++)
        {
                cout<<x<<endl;
        }
}
```

❖ Reference Variable :

- An ordinary variable is a variable that contains the value of some type. (int a=10)
- A pointer is a variable that stores the address of another variable. It can be dereferenced to retrieve the value to which this pointer points to. (int *b=&a)
- There is another variable that C++ supports, i.e., references. It is a variable that behaves as an alias for another variable.(int &b=a)

- When a variable is declared as a reference, it becomes an alternative name for an existing variable.
- A variable can be declared as a reference by putting '&' in the declaration.
- Also, we can define a reference variable as a type of variable that can act as a reference to another variable.

- '&' is used for signifying the address of a variable or any memory.
- Variables associated with reference variables can be accessed either by its name or by the reference variable associated with it.
- **Syntax:**

  data_type &ref = variable;
- **Example :**

```
int x = 10;
// ref is a reference to x.
int& ref = x;
// Value of x is now changed to 20
ref = 20;
cout << "x = " << x << '\n';
// Value of x is now changed to 30
x = 30;
cout << "ref = " << ref << '\n';
```

❖ Operators in C++ :

- **Assignment operator (=,+=,-=,*=,etc..)**
- The assignment operator assigns a value to a variable.
- **Arithmetic operators (+, -, *, /, %)**
- Operations of addition, subtraction, multiplication and division correspond literally to their respective mathematical operators. The last one, *modulo operator, represented by a percentage sign (%), gives the remainder of a division of two values*.
- **Increment and decrement (++, --)**
- The increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively.

- **Relational and comparison operators (==, !=, >, <, >=, <=)**
- Two expressions can be compared using relational and equality operators. For example, to know if two values are equal or if one is greater than the other.
- **Logical operators (!, &&, ||)**
- The operator ! is the C++ operator for the Boolean operation NOT. It has only one operand, to its right, and inverts it, producing false if its operand is true, and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand.
- **Conditional ternary operator (?:)**
- The conditional operator evaluates an expression, returning one value if that expression evaluates to
- true, and a different one if the expression evaluates as false.
  syntax :
- condition ? result1 : result2
- **Bitwise operators (&, |, ^, ~, <<, >>)**
- Bitwise operators modify variables considering the bit patterns that represent the values they store.

- **Special Operators :**
- C++ language supports many special operators too. They are used for some special processes in C++ programming.
- The special operators used in C++ programming language are,
- Comma (,)
- Pointer Operators (& and *)
- Member Selection (. and ->)
- sizeof( ) operator

- Let's Have Look Some Other important Operators in C++ :
    1. scope resolution operator
    2. member referencing operator
    3. memory management operator
    4. manipulators

❖ 1. Scope Resolution Operators :

- In a program, we can have same variable names in different blocks.
- The variable of outer block can be access by inner block.
- The variables declared in outer block are known as the global variable and the variables declared in the inner block are referred as local variables.
- Thus the local and global variables define their scopes regarding how long they are visible.
- You can use scope resolution operator to access the global variable in C++.

- **Syntax :**
- ::variable_name;
- **Example :**

```
int a=111;
void main( )
{
        int a=10;
        cout<<"Local A ="<<a;                    //10
        cout<<"Global A ="<<::a;                 //111
}
```

- **Uses of the scope resolution Operator :**
- It is used to access the hidden variables or member functions of a program.
- It defines the member function outside of the class using the scope resolution.
- It is used to access the static variable and static function of a class.
- The scope resolution operator is used to override function in the Inheritance.

➢ Program to define the member function outside of the class using the scope resolution (::) operator

- **Example :**

```
#include<conio.h>
#include<iostream.h>
   class operate
   {
         public:
         void fun();
   };
   void operate::fun()
   {
         cout<<"this is fun()";
   }
   void main()
   {
         clrscr();
         operate op;
         op.fun();
         getch();
   }
```

❖ 2. Member Referencing Operators :

Operator	Meaning
.	To access a member with object name and member name
->	To access a member with pointer to object and member name
.*	To access a member with object name and pointer to member
->*	To access a member with pointer to object and pointer to member
::*	To declare a pointer to member of a class

- Once a class is defined, its members can be accessed using two Operators :-

1. (.) Dot Operator,
2. (->)Arrow Operator

- While , (.) Dot Operator takes class or struct type Variable as Operand.
- (->) Arrow Operator takes a Pointer or Reference Variable as its Operand.

- **Example of (.) Operator :**

```
#include <iostream.h>
#include<conio.h>

class Student{
        public :
        int rollno , houseno;
};
void main ()
{
        Student s_obj;
         s_obj.rollno = 26 ;
         s_obj.houseno = 24 ;

        cout << "\nRoll no of the Student is = " << s_obj.rollno ;
        cout << "\nHouse of the Student is = " <<s_obj.houseno;
        getch();
}
```

- **Example of (->) Operator :**

```
#include <iostream.h>
#include<conio.h>
class Student{
        public :
        int rollno , houseno;
};
void main ()
{
        Student  s_obj;
        Student  *ptr_s_obj;
        ptr_s_obj=&s_obj;
         ptr_s_obj ->rollno = 26 ;
         ptr_s_obj -> houseno = 24 ;
        cout << "\nRoll no of the Student is = " << s_obj.rollno ;
        cout << "\nHouse of the Student is = " <<s_obj.houseno;
        cout << "\nRoll no of the Student is Using Pointer = " <<ptr_ s_obj->rollno ;
        cout << "\nHouse of the Student is = " << ptr_ s_obj->houseno;
        getch();
}
```

Other Example of all Operators :

```
#include<iostream.h>
#include<conio.h>
class demo
{
    public :
            int x;
            void fun(int a)
            {
                        x=a;
            }
};

void main()
{
    clrscr();
    demo obj;
    obj.fun(5);
    cout<<"with obj name & member
    name\n";
    cout<<"obj.x\n"<<obj.x<<“\n\n”;
    //->
    demo *ptr_obj=&obj;
    cout<<"pointer to object & member
    name";
    cout<<"\nptr_obj->x\n"<<ptr_obj->x
    <<“\n\n”;;

    //::*
    int demo::*p=&demo::x;
    cout<<"object name & pointer to
    member";
    cout<<"\nobj.*p\n"<<obj.*p<<"\n\n";

    //->*
    cout<<"pointer to object & pointer to
    member";
    cout<<"\nptr_obj->*p\n"<<ptr_obj->*p;

    getch();
}
```

❖ 3. Memory Management Operators :

- Dynamic memory allocation in C/C++ refers to performing memory allocation manually by a programmer. Dynamically allocated memory is allocated on **Heap,** and non-static and local variables get memory allocated on **Stack.**
- C++ also define two Unary Operators :
    1. **new**
    2. **delete**

- New and Delete Operators performs the task of allocating and freeing the memory. Since, these Operators manipulates memory on the Free Store, They are also known as **Free Store Operators.**

- **new operator :**
- The new operator denotes a request for memory allocation on the Free Store.
- If sufficient memory is available, a new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.
- **Syntax :**

  pointer-variable = **new** data-type;

  pointer-variable = **new** data-type[size]; //if array
- **Example :**

```
#include <iostream.h>
void main()
{
     // pointer to store the address returned by the new
     // allocating memory for integer
    int* ptr=new int;
    // assigning value using dereference operator
   *ptr = 10;
   // printing value and address
   cout << "Address: " << ptr << endl;
   cout << "Value: " << *ptr;
}
```

- **The following are the advantages of the new operator over malloc() function:**
- It does not use the sizeof() operator as it automatically computes the size of the data object.
- It automatically returns the correct data type pointer, so it does not need to use the typecasting.
- Like other operators, the new and delete operator can also be overloaded.
- It also allows you to initialize the data object while creating the memory space for the object.

- Examples of malloc vs new :
- ptr = (int*) malloc(100 * sizeof(int));
- int* ptr=new int;

- Delete operator :
- When memory is no longer required, then it needs to be deallocated so that the memory can be used for another purpose. This can be achieved by using the delete operator.
- **Syntax:**

  delete pointer-variable;
- **Example :**
- Delete with pointer :

  delete p;
  delete q;
- delete with an array :

  delete [ ]arr;

❖ 4. Manipulators :

- A manipulators are operators that are generally used for formatting the output or display the data in format.
- The most commonly used manipulator is **endl** and **setw**

Manipulators	**Description**
endl	It is used to enter a new line with a flush.
setw(a)	It is used to specify the width of the output.
setprecision(a)	It is used to set the precision of floating-point values.
setbase(a)	It is used to set the base value of a numerical number.

- **endl (end line)**
- The **endlmanipulator is used to insert a new line. It works similar to “\n”.**
- But as it is a manipulator, **it is not included in double quotes.**
- **Example :**

  cout<<“hello”<<“\n”<<“bca”;

  Same as :

  cout<<“hello”<<endl<<“bca”;

- Endl manipulator is under the #include<iostream.h>

- **setw (set width)**
- The setw manipulator is used to specify the width of variable to be printed in cout. **(Note :iomanip.h must include)**
- **Syntax :**
- setw(int width);
- **Example :**

```
#include<iomanip.h>
#include<iostream.h>
void main()
{
        int a=10, b=22020;
        cout<<"A="<<setw(6)<<a<<endl;
        cout<<"B="<<setw(6)<<b<<endl;
}
```

- Expressions can be one of the following :
- Integer Expressions
- Float Expressions
- Pointer Expressions
- Relational Expressions
- Logical Expressions
- Bitwise Expressions
- Constant Expressions

- ***Integer Expressions :***
- The integer expressions return the integer values after completing.
- **Example :**

```
int a=10, b=20, c;
c=a+b;
```

- ***Float Expression :***
- The float expression return the floating point results:
- **Example**

```
float a=1.5, b=1.75, c;
c=a+b*0.5;
```

- ***Pointer Expression :***
- The pointer expression returning address of some variable.
- **Example :**

```
int a, *p;
p=&a;
```

- **Relational Expression :**
- The relational expressions contain comparisons or conditions and return true or false type values.
- **Example :**

```
if(a>b)
```

- **Logical Expression :**
- The logical expressions combining relational expressions.
- **Example :**

```
if(a>b && a>c)
```

- **Bitwise Expression :**
- The bitwise expressions containing bitwise operators.
- **Example :**

```
a=x>>2;
x=a & b;
```

- **Constant Expressions :**
- The constant expressions containing only constant values.
- **Example :**

  a=2+4*5/2;
- **Special Assignment Expressions :**
- You can use more than one assignment operator in an expression to produce a chained is known as assignment expressions.
- **Example :**

  a=b=c=10;

Precedence of Operator :

- Operator precedence plays an important role in an expression when there are different operators are used.
- **Example :**

  ans=10+20*3+6/2-1

Type Casting :

- A type cast is basically a conversion from one type to another. There are two types of type conversion:
- **Implicit Type Conversion** Also known as 'automatic type conversion'.
  - Done by the compiler on its own, without any external trigger from the user.
  - Generally takes place when in an expression more than one data type is present. In such condition type conversion (type promotion) takes place to avoid lose of data.
  - All the data types of the variables are upgraded to the data type of the variable with largest data type.bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double
- It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long long is implicitly converted to float).

- **Example :**

```
#include <iostream.h>
#include<conio.h>
void main()
{
   int x = 10; // integer x
   char y = 'a'; // character c
   clrscr();
   // y implicitly converted to int. ASCII
   // value of 'a' is 97
   x = x + y;
   // x is implicitly converted to float
   float z = x + 1.0;
   cout << "x = " << x << endl
        << "y = " << y << endl
        << "z = " << z << endl;
   getch();
}
```

Explicit Type Conversion:

- This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.In C++, it can be done by two ways:
- **Converting by assignment:** This is done by explicitly defining the required type in front of the expression in parenthesis. This can be also considered as forceful casting.**Syntax:**
- (type) expressionwhere *type* indicates the data type to which the final result is converted.

Example :

```
#include<iostream.h>
#include<conio.h>
void main()
{
  double x = 1.2;
  clrscr();
  // Explicit conversion from double to int
  int sum = (int)x + 1;
  cout << "Sum = " << sum;
  getch();
}
```

Select Control Structure :

- If Statement
- We can write conditional statement IF in different ways…
  - if (Wimple if Without Else Statement)
  - if….else
  - if….else if ladder
  - Nested if (if within if)
- Switch Statement

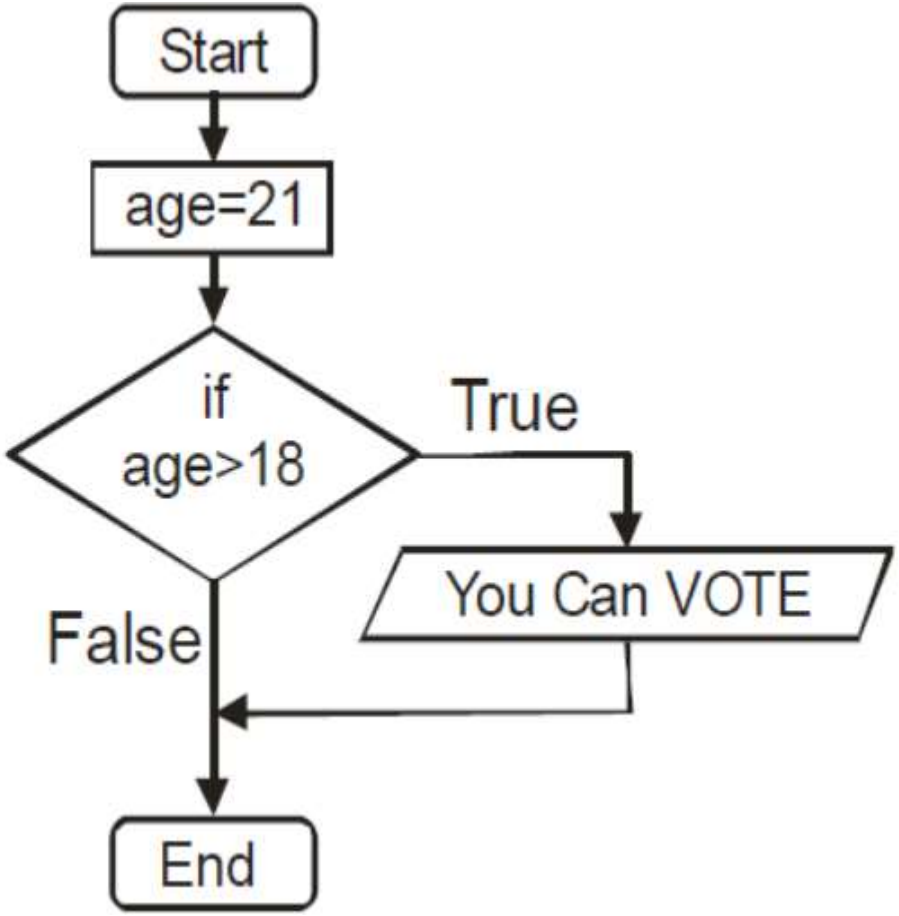
if (Simple if without else) :

- If statement is the decision control statement.
- This statement will check the given condition.
- *If condition is TURE then it execute given statements.*
- **Syntax :**

```
if(condition)
{
        statements;
}
```

- **Example :**

```
void main( )
{
   int age;
   cout<<“Enter Your Age : “;
   cin>>age;
   if(age>18)
        cout<<“You can VOTE”;
}
```



If-else :

- If statement is the decision control statement.
- This statement will check the given condition.
- If condition is TURE then it execute given statement or execute other statement.
- **Example :**

```
void main( )
{
        int age;
        cin<<“Enter Your Age : “;
        cout>>age;
        if(age>18)
                cout<<“You can VOTE”;
        else
                cout<<“You can not VOTE”;
}
```

if….else if ladder :

- You can use else…if ladder to execute a code based on multiple conditions.
- Here each condition is tested one after another.
- If condition is true then it execute the statement block and terminate the ladder.
- **Syntax :**

```
if(condition)
        statement block1;
else if(condition)
        statement block2;
else
        statement block3;
```

Nested if statement :

- You can nest multiple if blocks to test the one condition based on another condition.
- If condition1 is true then it checks the second condition.
- **Syntax :**

```
if(condition1)
{
        if(condition2)
        {
                //statement block1
        }
}
else
{
        //statement block2
}
```

Switch Statement :

- When we have relatively more number of conditions at that time we can use switch.
- **Syntax :**

```
switch(variable or expression)
{
        case value 1:
                statements;
                break;
        .......
        case value N:
                statements;
                break;
        default:
                default  statements;
}
```

Looping Statements :

- Loop means…
- In computer **programming, a loopis a sequence of instructions that is continue repeated until a specified condition is satisfied.**
- Loop is a concept to reduce repeated code.
- Looping can be divided in two type :

1. Entry Control Loop
    1. **for**
    2. **while**
2. Exit Control Loop
    1. **do…while**

For loop :

- For loop is the simplest loop and becomes very famous.
- It is entry-controlled loop, because the condition is tested before entering the block.
- **Syntax :**

```
for(initialization;condition;incr/decrement)
{
        //Block of Statements
}
```

While loop :

- While is a simple loop which just tests a condition and if the condition is true then block executed again.
- While loop is an entry-controlled loop as the condition is checked at the time of entering the block.
- **Syntax :**

```
while(condition)
{
    //Block of Statements
}
```

do...while Loop :

- Do while loop is an exit-controlled loop. It first executes the block and then tests condition.
- If the condition is true, the block of code is executed again.
- **Syntax :**

```
do
{
        //Block of Statements
} while(condition);
```

- More about applications of oop :
- https://www.geeksforgeeks.org/application-of-oops-in-cpp/
- More about C++ Tokens :
- https://www.geeksforgeeks.org/cpp-tokens/

Assignment Questions :

1. Write down history of c++.
2. Write top-down approach.
3. Write bottom-up approach.
4. Write down note on OOP in detail.
5. Write down application on c++.
6. Explain input / output operators in detail.
7. Write don short note on type compatibility.
8. Write don short note on reference variables.
9. Write a program to print following output :

```
*
* 2
* 2 *
* 2 * 4
```

```
1
1 *
1 * 3
1 * 3 *
```

10. Operators in C++:
    1. **scope resolution operator,**
    2. **member referencing operator,**
    3. **memory management operator,**
    4. **manipulators**