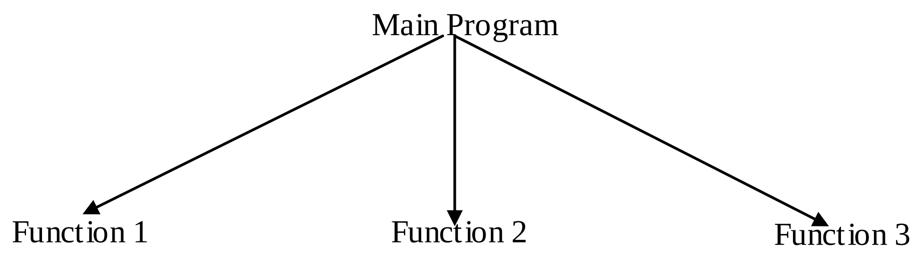
UNIT – 1 : Principles of object oriented programming tokens, expressions and control statements

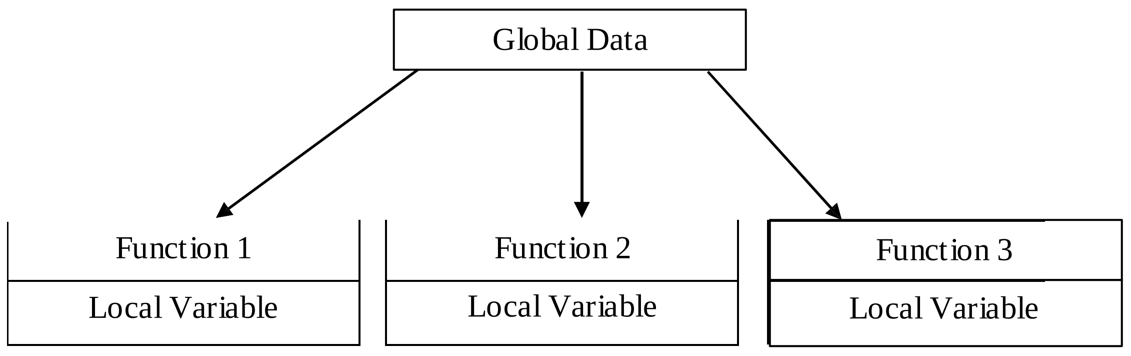
Topic: procedure-oriented programming (POP)

- Conventional programming using high level languages such as COBOL, FORTRAN and C, is commonly known as **procedure-oriented programming (POP)**.
- In the procedure-oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing.
- A number of functions are written to complete these tasks.
- The primary focus is on functions. A typical program structure for procedural programming is shown in fig.



Procedure-oriented programming basically consists of writing of list of instructions for the computer to follow

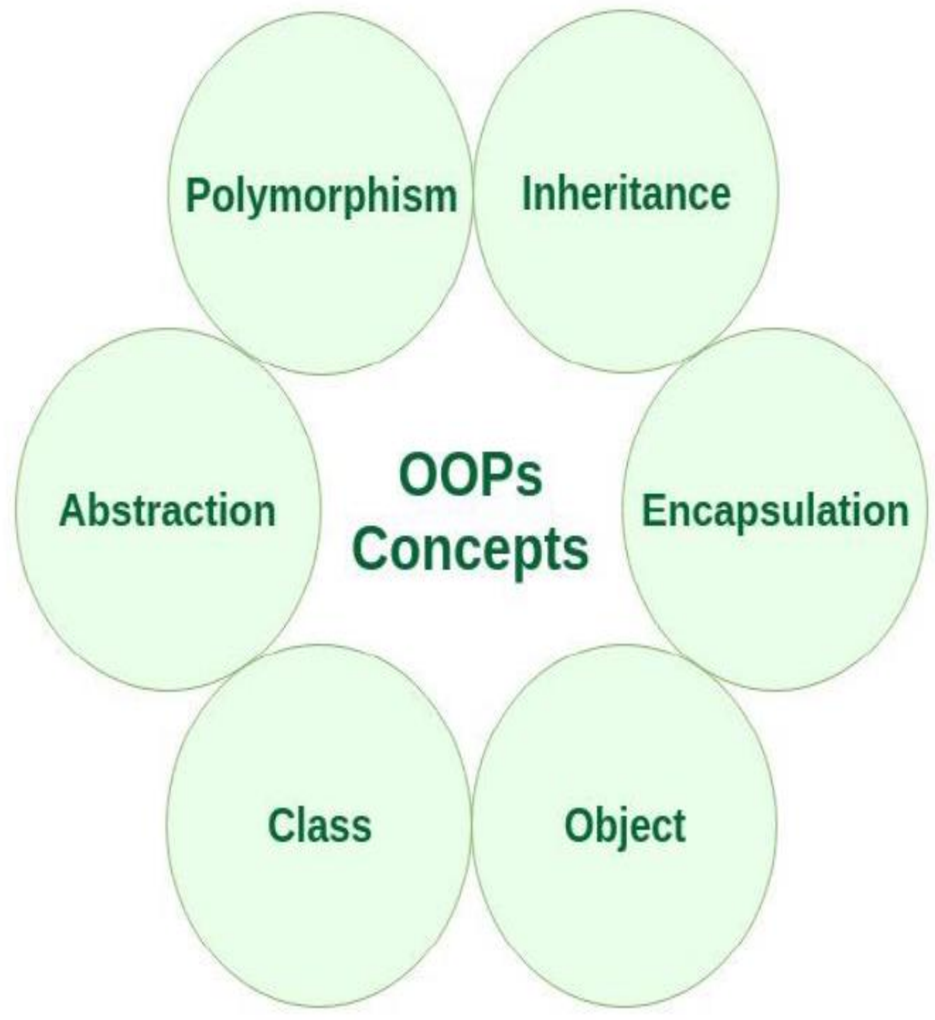
- This instructions groups known as functions.
- We normally use a flow chart to organize these actions and represent the flow of control from one action to another.
- In a multi –function program, many important data items are placed as global so that they may be accessed by all the functions .
- Each function may have its own local data.



Topic: Object -oriented programming (OOP)

- Object Oriented Programming is a programming in which we design and develop our application or program based of object. Objects are instances(variables) of class.
- Object oriented programming does not allow data to flow freely around the system. It binds data more closely to the functions that operate on it, and protects it from accidental modifications from outside functions.

Topic: Basic Concept of OOP



1. **Class** :

- Classes are an expanded version of structures. Structure can contain multiple variables.
- Classes can contain multiple variables, even more, classes can also contain functions as class member.
- Variables available in class are called Data Members.
- Functions available in class are called Member Functions.

2. **Object** :

- Class is a user-defined data type and object is a variable of class type.
- Object is used to access class members.

- Ex Of Class & Object
- class MyClass // The class

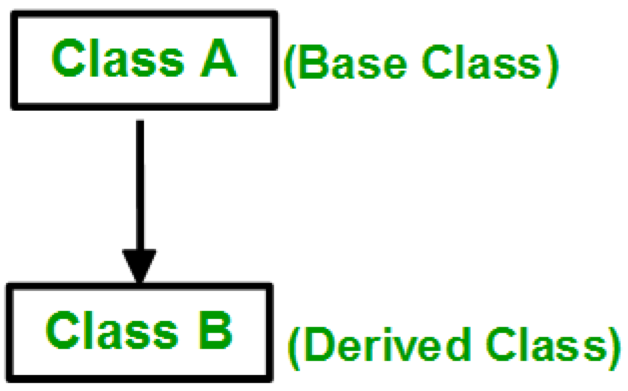
```
{
  public:          // Access specifie
   int myNum;       // Attribute (int variable)
   string myString;  // Attribute (string variable)
};

 int main()
 {
  MyClass myObj;  // Create an object of MyClass
  // Access attributes and set values
  myObj.myNum = 15;
  myObj.myString = "Some text";

}
```

3. **Inheritance** :

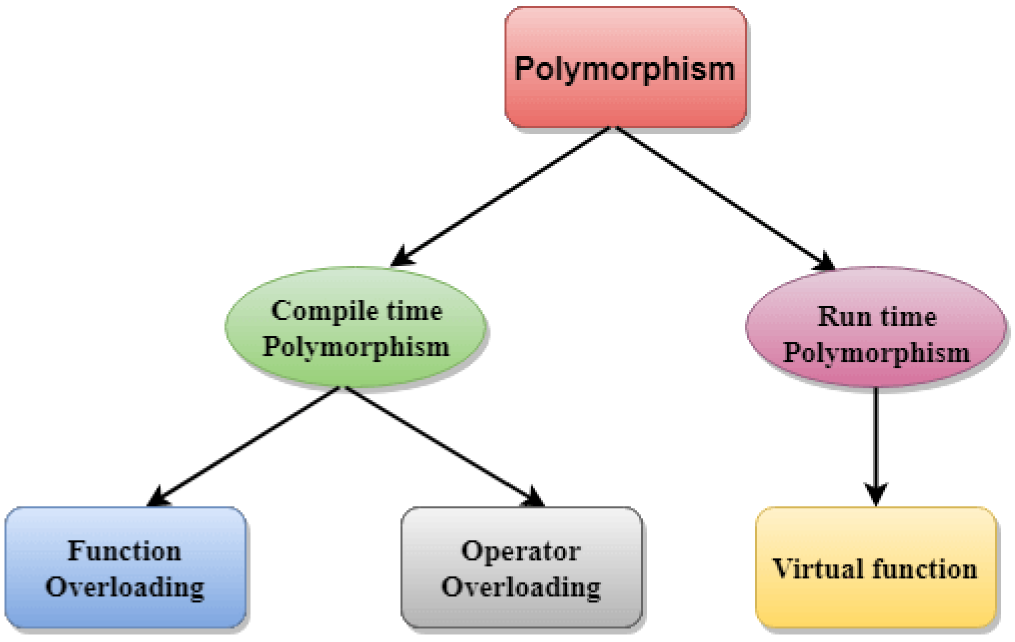
- Inheritance means access the properties and features of one class into another class.
- The class who is going to provide its features to another class will be called base class and the class who is using the properties and features of another class will be called derived class.



4. **Polymorphism** :

- Polymorphism means more than one function with same name, with different working. It can be static or dynamic.
- In static polymorphism memory will be allocated at compile time. In dynamic polymorphism memory will be allocated at runtime.
- Both function overloading and operator overloading are an examples of static polymorphism.

- Virtual function is an example of dynamic polymorphism.

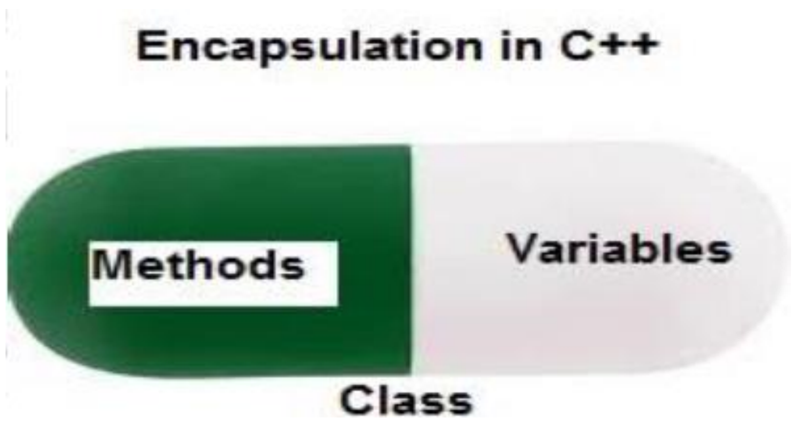


5. **Data Abstraction** :

- The basic idea of data abstraction is to visible only the necessary information, unnecessary information will be hidden from the outside world.
- This can be done by making class members as private members of class.
- Private members can be accessed only within the same class where they are declared.

6. **Encapsulation** :

- Encapsulation is a process of wrapping data members and member functions in a single unit called class.
- Using the method of encapsulation, the programmer cannot directly access the data.
- Data is only accessible through the object of the class.



Topic: Benefits of OOP's

- Through inheritance, we can eliminate redundant code and extend the use of existing classes which is not possible in procedure oriented approach.
- We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch which happens procedure oriented approach. This leads to saving of development time and higher productivity.
- The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
- It is possible to have multiple instances of object to co-exist without any interference.
- It is possible to map objects in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects .
- The data-centered design approach enables us to capture more details of a model in implementable from.
- Object oriented systems can be easily upgraded from small to large systems.
- Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

Topic :Object Oriented vs Procedure Oriented Programming

Procedure Oriented Programming	**Object Oriented Programming**
In POP, program is divided into small parts called functions.	OOP, program is divided into parts called objects.
In POP,Importance is not given to data but to functions as well as sequence of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a real world.
POP follows Top Down approach.	OOP follows Bottom Up approach.
POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
In POP, Most function uses Global data for sharing that can be accessed freely from function to function.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
POP does not have any proper way for hiding data so it is less secure.	OOP provides Data Hiding so provides more security.
Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Topic: Application of OOP & C++

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.

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

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

Topic: What is c++?

- Over the years, computer programs have become larger and more complex.
- Even though C is an excellent programming language, it has its limits. In C, once a program exceeds from thousands lines of code, it becomes so complex that it is difficult to maintain as a totality.
- The purpose of C++ is to allow this barrier to be broken. The essence of C++ is to allow the programmer to comprehend and manage larger, more complex programs.
- C++ began as an expanded version of C.
- The C++ were first invented by Bjarne Stroustrup in 1979 at Bell Laboratories in Murray Hill, New Jersey. This new language was initially called "C with Classes".

Topic: Input/Output operator

- In C++, input and output (I/O) operators are used to take input and display output.
- The operator used for taking the input is known as the extraction or get from operator (>>), while the operator used for displaying the output is known as the insertion or put to operator (<<).
- **Input Operator**
- The input operator, commonly known as the extraction operator (>>), is used with the standard input stream, cin. As stated earlier, cin treats data as a stream of characters. These characters flow from cin to the program through the input operator. The input operator works on two operands, namely, the c in stream on its left and a variable on its right. Thus, the input operator takes (extracts) the value through cin and stores it in the variable.

- To understand the concept of an input operator, consider this example.

```
int main ()
{
int a;
cin>>a; a
= a+1;
return 0;
}
```

- In this example, the statement cin>> a takes an input from the user and stores it in the variable a.

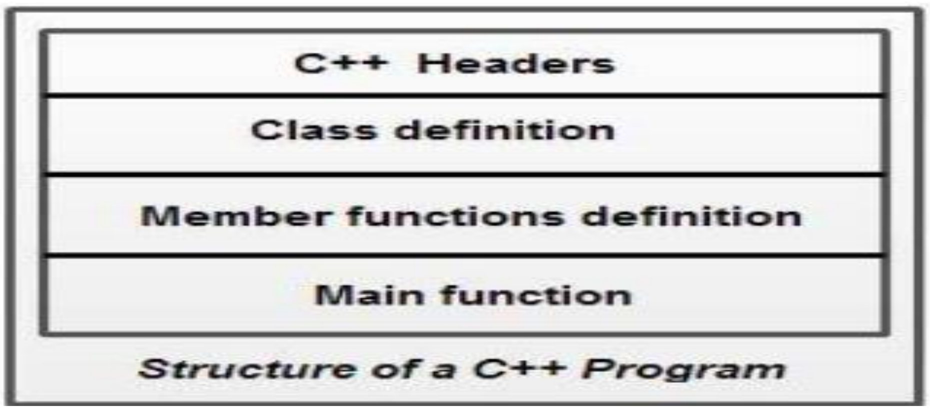
- **Output Operator**
- The output operator, commonly known as the insertion operator (<<), is used. The standard output stream cout Like cin, cout also treats data as a stream of characters. These characters flow from the program to cout through the output operator. The output operator works on two operands, namely, the cout stream on its left and the expression to be displayed on its right. The output operator directs (inserts) the value to cout.
- To understand the concept of output operator, consider this example.

```
#include<iostream>
int main ()
{
    int a;
    cin>>a;
    a=a+1;
    cout<<a;
    return 0;
}
```

- This example is similar to Example 1. The only difference is that the value of the variable a is displayed through the instruction cout << a .

Topic: Structure of a C+ + Program

- 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.
- C++ supports two comment styles: single line comment and multiline comment. Single line comments are used to define line-by-line descriptions. Double slash (//) is used to represent single line comments. To understand the concept of single line comment, consider this statement.
- / / An example to demonstrate single line comment It can also be written as
- / / An example to demonstrate
- / / single line comment
- Multiline comments are used to define multiple lines descriptions and are represented as / * * /. For example, consider this statement.

- /* An example to demonstrate
- multiline comment */
- Generally, multiline comments are not used in C++ as they require more space on the line. However, they are useful within the program statements where single line comments cannot be used.
- **Headers**: 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. The standard headers contain the information like prototype, definition and return type of library functions, data type of constants, etc. As a result, programmers do not need to explicitly declare (or define) the predefined programming elements.

- Standard headers are specified in a program through the preprocessor directive" #include. In Figure, the iostream header is used. When the compiler processes the instruction #inc1ude<iostream>, it includes the contents of iostream in the program. This enables the programmer to use standard input, output and error facilities that are provided only through the standard streams defined in <iostream>. These standard streams process data as a stream of characters, that is, data is read and displayed in a continuous flow. The standard streams defined in <iostream> are listed here.
- cin (pronounced "see in") : It is the standard input stream that is associated with the standard input device (keyboard) and is used to take the input from users.
- cout (pronounced "see out") : It is the standard output stream that is associated with the standard output device (monitor) and is used to display the output to users.
- **Namespace:** Since its creation, C++ has gone through many changes by the C++ Standards Committee. One of the new features added to this language is namespace. A namespace permits grouping of various entities like classes, objects, functions and various C++ tokens, etc., under a single name. Different users can create separate namespaces and thus can use similar names of the entities. This avoids compile-time error that may exist due to identical-name conflicts.
- The C++ Standards Committee has rearranged the entities of the standard library under a namespace called std. In Figure, the statement using namespace std informs the compiler to include all the entities present in the namespace std. The entities of a namespace can be accessed in different ways which are listed here.

- By specifying the using directive
- **using namespace std;**
- **cout<<"Hello World";**
- By specifying the full member name
- **std: :cout<<"Hello World";**
- By specifying the using declaration
- **using std:: cout;**
- **cout<<"Hello World";**

- As soon as the new-style header is included, its contents are included in the std namespace. Thus, all the modern C++ compilers support these statements.
- **#include<iostream>**
- **using namespace std;**
- However, some old compilers may not support these statements. In that case, the statements are replaced by this single statement.
- **#include<iostream.h>**

- **Main Function**: The main () is a startup function that starts the execution of a c++ program. All C++ statements that need to be executed are written within main ( ). The compiler executes all the instructions written within the opening and closing curly braces' {}' that enclose the body of main ( ). Once all the instructions in main () are executed, the control passes out of main ( ), terminating the entire program and returning a value to the operating system.
- By default, main () in C++ returns an int value to the operating system. Therefore, main () should end with the return 0 statement. A return value zero indicates success and a non-zero value indicates failure or error.

Topic: What is Namespace?

- Consider following C++ program.
- // A program to demonstrate need of

```
namespace int main()
{
   int value;
   value = 0;
   double value; // Error
   here value = 0.0;
}
```

- Output :
- Compiler Error:
- 'value' has a previous declaration as 'int value'
- In each scope, a name can only represent one entity. So, there cannot be two variables with the same name in the same scope. Using namespaces, we can create two
  variables or member functions having the same name.

```
// Here we can see that more than one variables
// are being used without reporting any error.
// That is because they are declared in the
// different namespaces and scopes.
#include <iostream>
using namespace std;

// Variable created inside
namespace namespace first
{
   int val = 500;
}
```

```
    // Global variable
int val = 100;

int main()
{
   // Local variable
   int val = 200;

   // These variables can be accessed from
   // outside the namespace using the scope
   // operator ::
   cout << first::val << '\n';

   return 0;
}
```

- Output:
- 500

- Definition and Creation:
- Namespaces allow us to group named entities that otherwise would have global scope into narrower scopes, giving them namespace scope. This allows organizing the elements of programs into different logical scopes referred to by names.
- Namespace is a feature added in C++ and not present in C.
- A namespace is a declarative region that provides a scope to the identifiers (names of the types, function, variables etc) inside it.
- Multiple namespace blocks with the same name are allowed. All declarations within those blocks are declared in the named scope.

Topic: Tokens in C++

- Tokens is the smallest individual unit in C++ language. In fact, every unit that makes a sentence in C++ is a Token. C++ has ten types of Tokens as given below.
    1) Keywords
    2) Identifiers
    3) Basic data types
    4) User-derined types
    5) Derived data types
    6) Symbolic constants
    7) Type compatibility
    8) Declaration of variables
    9) Dynamic initialization of variables
    10) Reference variables

1) Keywords:

- Keywords are those words who has special meaning for compiler. We can't use keywords as variable name.
    - C++ has 32 Keywords as follows:

Keywords			
Auto	double	int	struct
Break	else	long	switch
Case	enum	register	typedef
Char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
Default	goto	sizeof	volatile
Do	if	static	while

2) Identifiers

- Identifiers are fundamental building blocks of a program and are used as the general terminology for the names given to different parts of program viz. variables, functions array structures etc.

3) Basic data types

Type	Typical Bit Width	Typical Range
Char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
Int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,648 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
Float	4bytes	+/- 3.4e +/- 38 (~7 digits)
Double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)

4) User-defined types

- The data types that are defined by the user are called the derived datatype or user-defined derived data type.
  These types include:

1.) Class

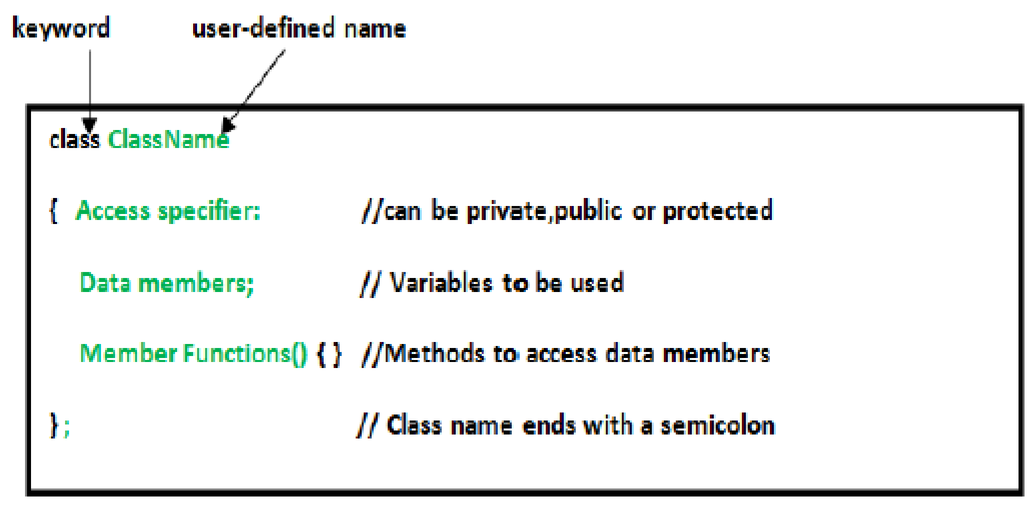
2.) Structure

3.) Union

4.) Enumeration

Class:

- The building block of C++ that leads to Object-Oriented programming is a Class. It is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.
- **Syntax**:



```
class ClassName
{  Access specifier:          //can be private,public or protected
   Data members;              // Variables to be used
   Member Functions() { }     //Methods to access data members
};                            // Class name ends with a semicolon
```

Srtucture:

- A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type.

Syntax:

```
struct address {
   char
   name[50];
   char
   street[100];
   char city[50];
   char
   state[20];
    int
   pin;
};
```

Union:

- Like Structures, union is a user defined data type. In union, all members share the same memory location. For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

Enumeration:

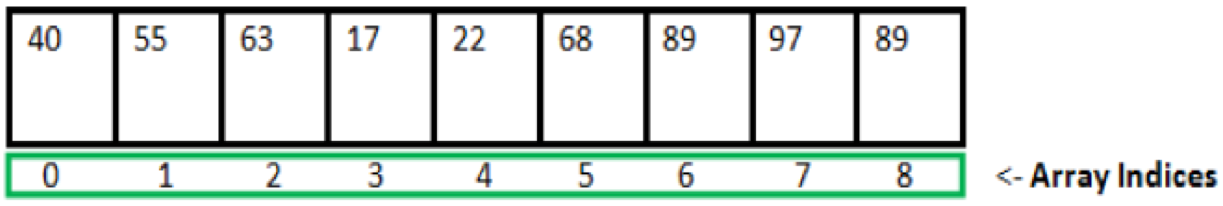
- Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.
- **Syntax:**
- enum State {Working = 1, Failed = 0};

5) Derived data types

- The data-types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. These can be of four types namely:
  1.) Function
  2.) Array
  3.) Pointers
- Let's briefly understand each of the following derived datatypes:

1. **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:**
     FunctionType FunctionName(parameters)

2. **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.



Array Length = 9
First Index = 0
Last Index = 8

   - **Syntax:**
   DataType ArrayName[size_of_array];

3. **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. It's
     general declaration in C/C++ has the format:

- **Syntax:**
  datatype *var_name;

6) Symbolic constants

- Any value declared as a const can not be modified by the program in any way.
- Syntax: const variable-name=value;

```
Ex: #include<iostream.h>
    #include<conio.h>
    #include<iomanip.h>
   Void main()
   {
       Const i=10;
       Cout<<i;
       I=i+1; //generate an error because of constant value can't modify
       getch();
   }
```

7) Type compatibility

- C++ is very strict with regard to type compatibility as compared to C. Type compatibility is very close to automatic or implicit type conversion.
- The type compatibility is being able to use two types together without modification and being able to subsititute one for the other without modification.
- **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 of Type Implicit Conversion:

```
#include <iostream>
using namespace std;

int main()
{
  int x = 10; // integer x char y = 'a'; // character c

  // 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;

  return 0;
}
```

Output:

```
x = 107
y = a
z = 108
```

- **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) expression

  Example:

```
// C++ program to demonstrate
// explicit type casting

#include <iostream>
using namespace std;

int main()
{
   double x = 1.2;

   // Explicit conversion from double to int
   int sum = (int)x + 1;

   cout << "Sum = " << sum;

   return 0;
}
```

  Output
  : Sum =
  2

- **Advantages of Type Conversion:**
- This is done to take advantage of certain features of type hierarchies or type representations.
- It helps to compute expressions containing variables of different data types.

8) 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, olny 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 of Variable
- Declaration will allocate memory for specified variable with garbage value.
- Syntax :

```
Data-Type Variable-name;
```

- Examples :

```
int a;
float
b;
char c;
```

- Initialization means assigning value to declared variable. Every value will overwrite the previous value.
- Examples :

```
a = 10;
b = 4.5;
c = 'a';
```

- Character value must be enclosed with single
  quotes. a = 4.5;
- if we assign decimal value to integer variable, it will accept only integer portion of value. In the above example variable a will accept 4 only.

9) Dynamic initialization of variables

- Here, the variable is assigned a value at the run time. The value of this variable can be altered every time the program is being run.

10) Reference variables

- Think of a variable name as a label attached to the variable's location in memory. You can then think of a reference as a second label attached to that memory location. Therefore, you can access the contents of the variable through either the original variable name or the reference. For example, suppose we have the following example

```
int i = 17;
```

- We can declare reference variables for i as follows.

```
int& r = i;
```

- Read the & in these declarations as reference. Thus, read the first declaration as "r is an integer reference initialized to i" and read the second declaration as "s is a double reference initialized to d."

Topic: What is Operators? Explain types of Operators.

- An operator is a symbol that tells the compiler to perform specific mathematical or logical calculations on operands(variables).
- **Scope Resolution Operator**
- C++ is also a Block-Structured Language. The Scope of a variable extends from the point of its Declaration till the end of the code block, containing the declarations. A Variable
  declared inside a code block is said to be local to that code block. ::(Scope Resolution Operator) Operator allows access to the global version of a Variable.
  1) Global Scope Resolution Operator :
  Let us see an program Example illustrating the use of Scope Resolution Operator (::) used with the Gloal Variable is given below :

```
#include <iostream>
using namespace std;

int Sum = 10;

int main ()
{
   int Sum = 50;
   cout << "Value of local Sum variable in the main function = " << Sum ;
   cout << "\nValue of Sum using Scope Resolution Operator = " << ::Sum ;
   return 0 ;
}
```

2) Class Scope Resolution Operator :
In the below example we are using Scope Resolution Operator to define the class functions outside the class :

```
#include <iostream>
using namespace std;

class Scope_Resolution_Operator
{
   public :
            void Display();
}
void Scope_Resolution_Operator :: Display
{
   cout << "We are in the Display Function." ;
}
int main ()
{
   Scope_Resolution_Operator Scopeobj;
   Scopeobj.Display();
   return 0 ;
}
```

- **Member Dereferencing Operators**
- Once a class is defined, its members can be accessed using two Operators :-
  1) (.) Dot Operator, And
  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.
  Let us see an simple example of using (.) Dot operator to make access to the members defination of structure Student :

```
#include <iostream>
using namespace std;

struct Student{
   string name , state;
   int rollno , houseno;
};
int main ()
{
   Student amit;
   amit.name = "Amit" ;
   amit.rollno = 26 ;
   amit.houseno = 24 ;
   amit.state = "Delhi" ;

      cout << "Name of the Student is = " << amit.name ; cout
   << "\nRoll no of the Student is = " << amit.rollno ; cout <<
   "\nHouse of the Student is = " << amit.houseno ; cout <<
   "\nState of the Student is = " << amit.state ; return 0 ;
}
```

- *Let us see an simple example of using (->) Arrow operator to make access to the members defination of structure Student :*

```
#include <iostream>
using namespace std;

struct Student{
   string name , state;
   int rollno , houseno;
};
int main ()
{
   Student amit;
   Student *amitptr;
   // Here, *amitptr points to the object of Structure Student i.e amit

   amitptr = &amit;
   // Here, amitptr is equal to the address of the object amit

   amitptr -> name = "Amit" ;
   amitptr -> rollno = 26 ;
   amitptr -> houseno = 24 ;
   amitptr -> state = "Delhi" ;

   cout << "Name of the Student is = " << amit.name ; cout
   << "\nRoll no of the Student is = " << amit.rollno ; cout <<
   "\nHouse of the Student is = " << amit.houseno ; cout <<
   "\nState of the Student is = " << amit.state ; return 0 ;
}
```

- **Memory Management Operators**
  C++ also define two Unary Operators :-
  1) New, and
  2) Delete.

  - New and Delete Opeartors 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.**
  - A Data Object created inside a block with New Operator will remain in existence Until it is explicitly destroyed using Delete Operator.
  - When An Object Is No Longer Needed, It Is Destroyed To Release The Memory Space For Reuse by another variables by using Delete Operator.
  - *The General Syntax of using New Operator is :*
    Pointer_variable = New Data_type(Value);
    Let us see an simple example of using new operator :

```
#include <iostream>
using namespace std;

int main ()
{
        int *pointer;
        pointer = new int;
      // Now new operator dynamically allocates the memory space to the
pointer variable
        *pointer = 12;

        cout << *pointer ;
        return 0 ;
}
```

- The General Syntax of using Delete Operator is :
  Delete Pointer_variable;

  Let us see an simple example of using delete operator :

```
#include <iostream>
using namespace std;

int main ()
{
        int *pointer;
        pointer = new int;
        *pointer = 12;

        cout << *pointer ;
        // This returns the value 12 instead of the address value of pointer

        delete pointer;
        // delete operator deallocates the memory space of pointer and
now pointer variable does not contain any value

        cout << "\n" << *pointer ;
        // Now this will return the memory address of the pointer

        return 0 ;
}
```

- These operators are used to format the data display. Some of the manipulators are as follows :

- **1) endl manipulator -**
  - endl manipulator when used causes a linefeed. endl is a manipulator which is manipulating the actual nature of program by printing the next specified code in the next new line. endl does not get any parameters.
    For Example :-

```
#include<iostream>
using namespace std;

int main ()
{
        cout << "This is text of first line" << endl;
        cout << "This is text display in second line";
        // If you try to print this without endl the both lines will come in
oneline.

        return 0;
}
```

- **2. setw() manipulator -**
  setw(value) is a manipulator which sets the width before the character or can say that gives the number of space set in the parameter before the text is displayed. We can also fill the setted space by some character by the manipulator "setfill('character or value'). This manipulator takes the parameter which is a int datatype. To use this manipulator you must add "#include<iomanip>". This manipulator will only allow one character after its declaration.
  For Example :-

```
#include <iostream>
#include <iomanip>
using namespace std;

int main ()
{
        int a = 10;   // value to be putten in setw must be declared in a variable first

        cout << setw(a) << "Setw" << endl;
        cout << setw(a) << setfill('#') << "Setfill";

        return 0;
}
```

- **3. hex manipulator -**
- hex is a manipulator which manipulates the written expression and print the hexadecimal value of that expression. hex manipulator also does not take any parameters. For
  Example :-

```
#include<iostream>
//#include <iomanip>
using namespace std;

int main ()
{


       cout << hex << 50 << endl; cout <<
       hex << 100 << endl; cout << hex <<
       150 << endl; cout << hex << 200;
       // All values will be printed in their hexadecimal values by hex
   manipulator

       return 0;
   }
```

Topic: **Looping Structure**

- A Loop is defined as a block of processing steps repeated a certain no of times.
- **For Loop:**
- The For loop statement is used to repeat a statement or a block of statement specified number of times.
- **Syntax:**
- For(initialization; condition; updation)
- {
    - Body of loop
- }
- According to the above syntax the initialization part is initialized the variable.
- The condition part tested the loop variable if the condition is true, the body of the loop get executed else the loop is terminated and the next statement after the loop execut
  ed.
- In the updation part increments or decrements the loop variable. After testing condition update part is executed.
- The condition is check again and the whole process is repeated till the condition will false.
- **While**
- The while loop is used when the no of iterations to be performed are not known in advance.
- The statement in the loop is executed if the test condition is true and the execution continuous as long as it remains true.
    - **Syntax:**
    - While(condition)
    - {
        - Body of the loop
    - }
- According to the above syntax first execute the condition, if condition is true execute the body of loop, else loop will terminate.

- **Do….While**
- Sometimes it is required to execute the body of the loop at least once even if the test expression execute false during the first iteration.
- This requires testing termination expression at the end of loop rather than beginning.
- For that do….while loop is used where the test condition is at the bottom of the loop.
- This means that the program always execute the statements at least once.

- **Syntax:**
- Do
- {
    - Body of the loop
    - } while (condition);
- According to the syntax first execute the body of the loop and at last execute the condition and if condition is true then repeat body of loop once again, else control come out from the loop.

Topic: Explain the main Function.

- The execution of each and every c++ program is start from the main() function. It is the entry point of a program execution.
- The general format of main() function is as follow.
- **Syntax:**
- Return type main([int argc, char *argv[ ],[char ** envp]])
- {
    - Body of the main function
- }
- According to the above format the **return type** of the main() function must be either void or int.
- Here, return type specifies the status of the program termination.
- The main() function can also takes arguments from command prompt to. It is known as command line arguments.
- **argc** specifies the total number at argument. It is the argument counter. Its value is always positive.
- **Argv** represents the argument vector(array). It holds pointer to the argument passed from the command line.
- **argv[]** is one kind of an array so it holds the data in following manner:
    - argv[0] = pointer to the name of the executable program.
    - Argv[1], argv[2]….. argv[n] = pointers to argument strings
- **envp** represents an environment parameter. It is optional.

For example

```
#include<iostream.h>
#include<conio.h>
int main(int argc, char *argv[])
{
        int i;
        cout<<endl<<"Total arguments="<<argc;
        cout<<endl<<"Program name is="<<argv[0];
        cout<<endl<<"Other Arguments are";
        for(i=1;i<argc;i++)
        {
                cout<<endl<<argv[i];
        }
        getch();
        return(0);
}
```

run the above program from dos shell and enter following arguments:

```
C:\TC\BIN\SOURCE> ARGS.EXE I like c++ very much
```

Topic: Explain the call by reference.

- A reference as its name, is like alias.
- It refer to the same entity. A variable and its reference are tightly attached with each other.
- So, change in one it will also change in the other. When call any function by its reference any modifications made through the formal pointer parameter is also reflected in the actual parameter.
- It has functionality of pass-by-pointer and the syntax of call-by-value.
- In the function declaration parameter are to be received by reference must be preceded by the **& operator** and arguments or parameters pass same as call by value.
- However any modification in the variable in function body directly reflected to the actual parameter.
- Ex.
  - Void swapref(int &x, int &y);
- In above ex we can see that the reference of the variable will passed in the function parameters.
- When we want to call this type of function we can call same as call by value functions. Like **swapref(x,y).**

Topic: Explain the return by reference.

- We can pass a reference to a parameter same we can also return a reference from a function.
- A function that returns a reference variable is actually an alias for the referred variable.
- Here. A function can be called on the receiving side of an assignment.
- For ex.

```
          Int & max(int &x,int &y)
```

- In above ex you can see that the return type of function is int reference.

Topic: Explain the inline function.

- C++ provides an inline functions to reduce the function call overhead. Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call.
- When we call the function,control of program jumps to the code of the function, the CPU stores memory address of the instruction and copies arguments onto the stack, and finally transfers control to the specified function.
- CPU executes the function code, stores the return value in memory location or register and returns control to the calling function.
- If we call function 5 times, then the control of the program jumps to the set of instruction 5 times and perform same task to store argument in stock and return value to register again and again.
- C++ provides a solution to overcome such a problem. For that declare a function with the keyword inline. It is known as inline function.
- Here, the compiler does not create a one copy of real function, but copies the code where the function call have been made.
- If the function is called 5 times, the code of the function copied into the calling function each of the 5 times.
- It will improve the speed at the program but increase the size of executable program.
- The inline functions are defined as follow.
- Syntax:

```
        Inline returntype functionname()
        {
                //function body
        }
```

- According to above format the keyword inline is placed with function header at the time of function definition and also with function prototype.
- Some of the situation where inline function may not work are:
- If a loop, switch or goto exists.
- If returns statement exists but function not return a value.
- If a function contains static variables.
- If inline functions are recursive.

Topic: Explain default Argument.

- In c it a function is defined to receive 3 arguments, so whenever call a function need to pass value those 3 arguments to the function.
- It assigns a garbage value for argument for the last argument.
- In c++ functions have an ability to define default values for arguments that are not passed when the function is call.
- The default arguments can be specified by following the arguments is name with = default value in the function argument list.
- The default value must be specified from right-to-left specify a value to a particular argument in the middle of an argument list is not possible.
- A function can be declare with a default argument as follows:
    - Int sum(int x=5, int y=10, int z=15);
    - Int sum(int x, int y=10, int z=15);
    - Int sum(int x, int y, int z=15);
- Here, default arguments are specified from left to right.
- If specifies default argument value of any argument then also pass value for it, the new passed value is consider as a value of argument instead of default argument.

Topic: Explain Const argument.

- The keyword const specifies that the value of variable will not change throughout the program.
- If anyone attempt to after the value of variable defined with this qualifier an error can be created.
- A function can also take an argument as a const. which is specifies no any modification on the value.

Topic: Explain function overloading.

- Overloading refers to the use of a same thing for different purpose.
- It is the polymorphism feature of a object oriented concept.
- Function overloading or function polymorphism is a concept that allows multiple functions to use the same name with different number of argument and types of argument.
- In c language that can not possible to make any function with same name while in c++ it can be possible with function overloading or function name overloading.
- For ex:
    - Char * copy(char *);
    - int copy(int);
    - Float copy(int);
- Here, the above three function use the same function name copy() but work with different data types.
- Here, return type doesn't mean.
        - int copy(int,int);
        - Void copy(int,int);
- As shown in above two functions, it can not work,it will create ambiguity.
- Return type does not differentiate any function.
- When any function is called, the selection of the particular function is made from given forms at function and it done through the process of argument matching.
- The actual argument of the function call gets matched with the formal argument of each forms of the function.
- One of the following choice will occur when a call for the overloaded function is made.
- A match
- No match
- Ambiguous match

- Match
- Here, the compiler first tries to find exact match in which types of arguments are the same, and use that function.
- No match
- A no match occurs when the actual argument cannot be match with an argument of the defined function and will cause the error message.
- Ambiguous match
- If an exact match is not found. C++ compiler tries to match the arguments by two ways.
- A match through promotion.
- For ex:
    - Void ff(int);
    - Void ff(char *);
- Here, if function called as ff('a') . if is not exact match but 'a' is of type char.
- It is promoted to type int after no exact match is found.

- The other promotion such as.
- Float to double
- Enumeration to int
- Char to int
- A match by standard conversion.
- Here, compiler tries to use the built-in conversion to the actual argument and then uses the function whose match is unique.
- For ex:
- Void ff(char *c);
- Void ff(double d);
- Void ff(void);
- If any function called as
- ff(10); // Matches ff(double)
- ff(“a”);// match ff(char *)
- If the conversion have multiple matches, then the compiler will generate an error message.