Unit-1 Object Oriented Programming with C++

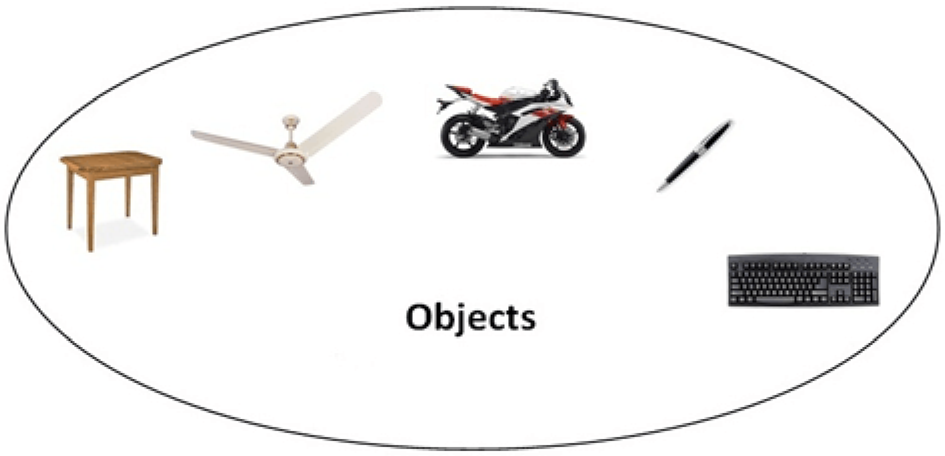
BCA SEM-3

Unit-1	Principal Of Object Oriented Programming	16	18
	- Introduction of OOP, OOP V/s POP - Concept of OOP – Object, Class, Inheritance, Encapsulation, Polymorphism, Abstraction ,Message Passing - Structure Of C++ Program - Tokens in C++ - Data type, Constant, Variable, Statement & Operators - Function – Member function, Inline function, Friend function - Input/output statements - Declaration & Creation of Class and Object		

BCA SEM-3
Assistant Professor: Vinod.M.Makwana
KBPCCS

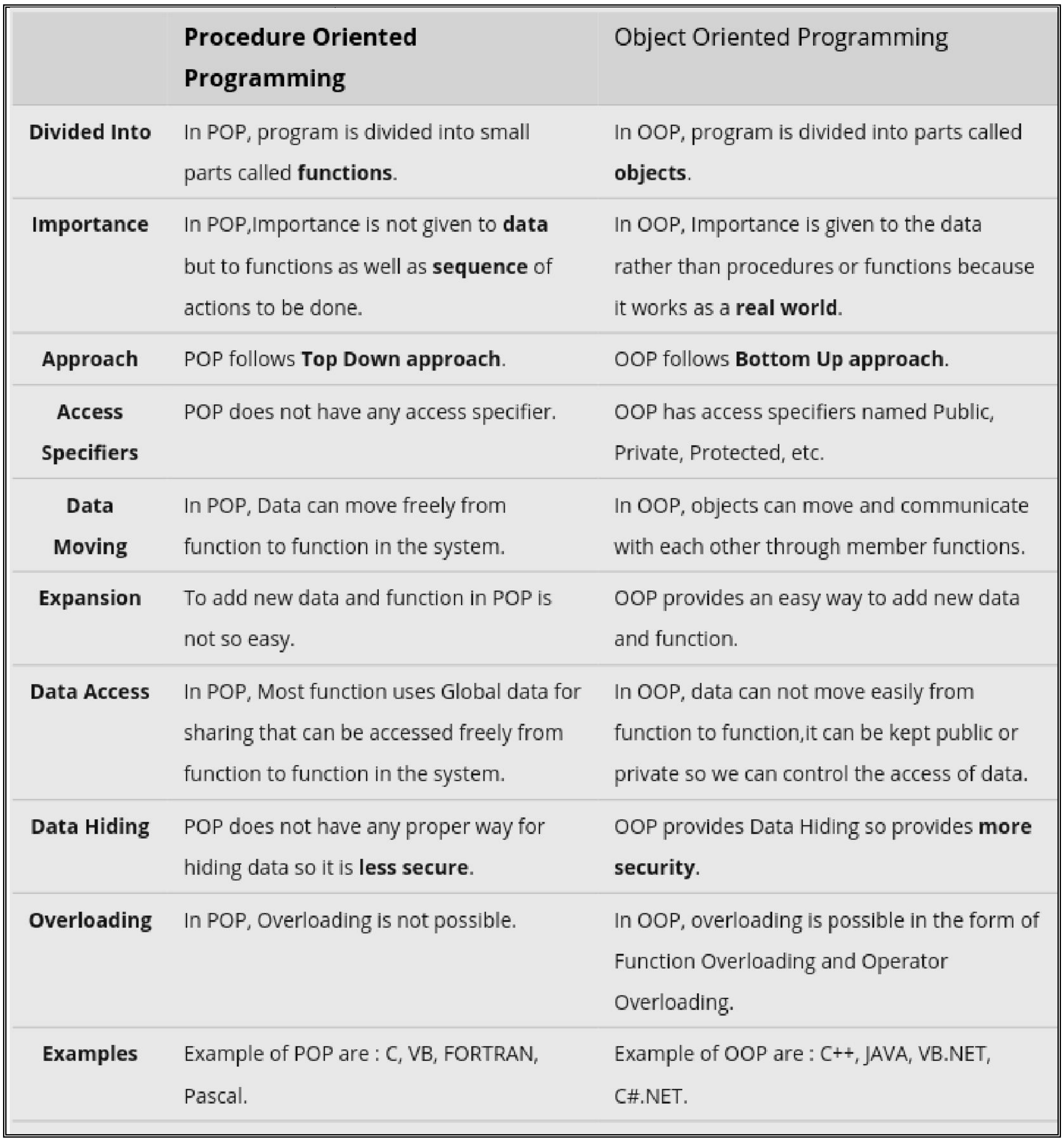
Que: 1 What is OOP? Give Differences between OOP and POP.

- **Object-Oriented Programming** is a methodology to design a program using classes and objects.
- **Class** is a logical entity which represents properties and behaviour, fruit, animal, furniture, bird are examples of class.
- **Object** means a real word entity such as pen, chair, table, mango, dog etc.
- The programming methodology where everything is represented as an object is known as truly object-oriented programming language.
- **Smalltalk** is considered as the first truly object-oriented programming language.
- It simplifies the software development and maintenance by providing some concepts:



- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Differences



	Procedure Oriented Programming	Object Oriented Programming
Divided Into	In POP, program is divided into small parts called **functions**.	In OOP, program is divided into parts called **objects**.
Importance	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**.
Approach	POP follows **Top Down approach**.	OOP follows **Bottom Up approach**.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	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.
Expansion	To add new data and function in POP is not so easy.	OOP provides an easy way to add new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	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.
Data Hiding	POP does not have any proper way for hiding data so it is **less secure**.	OOP provides Data Hiding so provides **more security**.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

Que: 2 what is object oriented programming? Explain its various concepts.

A programming language where everything is represents as class or object is called object orient programming.

Class represents data members and member function while object represents state and behaviour of the class.

[1]Class

- It is a user-defined data type, which holds its own data members and member functions, which can be accessed using its object only.
- A class is like a blueprint for an object.
- For Example: Consider the Class of Cars. There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A Class is a user-defined data-type which has data members and member functions.
- Data members are the data variables which are used to store some information where Member function is used to manipulate the data.
- It's a logical entity.
- In the above example of class Car, the data member will be speed limit, mileage etc and member functions can apply brakes, increase speed etc.

We can say that a **Class in C++** is a blue-print representing a group of objects which shares some common properties and behaviours.

Example

```
class Student
 {
    public:
          int id;  //field or data member
          float salary; //field or data member
          char name[20];//field or data member
 }
```

[2]Object

- In C++, Object is a real world entity, for example fruit is a class then mango, grapes, and apples are the objects.
- Object name can be anything; object is a physical entity which we can see it.
- In other words, object is an entity that has state and behaviour. Here, state means data and behaviour means functionality.
- Object is a runtime entity, it is created at runtime.
- Object is an instance of a class. All the members of the class can be accessed through object.
- It's a physical entity.

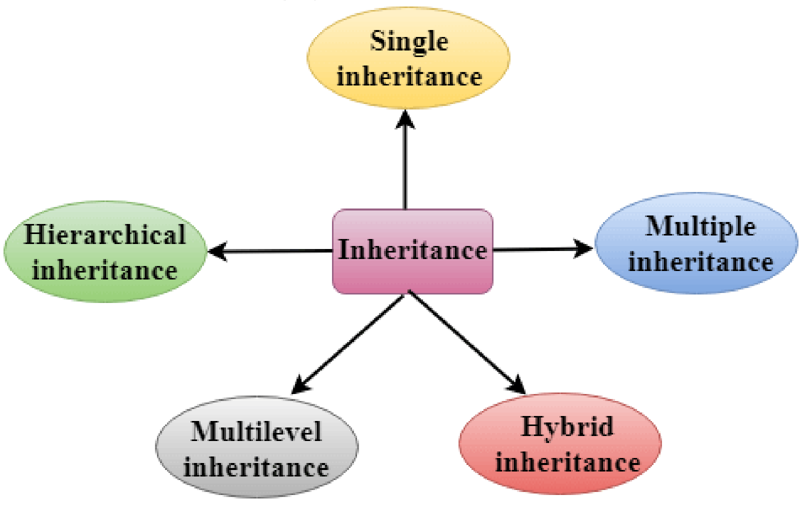
- Memory block is occupied when object is created.

Example

```
Student s1;  //creating an object of Student
```

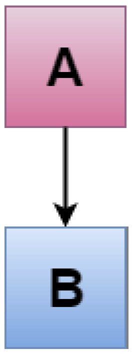
[3]Inheritance

- Acquiring the properties and behaviour of one class into another class is called inheritance.
- Re-usability of class is also known as inheritance.
- The class whose members are inherited is called base or parent class while the class which inherits the members of another class is called derived or child or sub class.
- C++ support following types of inheritance.



[A]Single level inheritance:

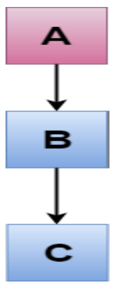
Inheritance in which a derived class is inherited from the only one base class is called single level inheritance.
Example:



Where 'A' is the base class and 'B' is the derived class.

[B] Multilevel inheritance

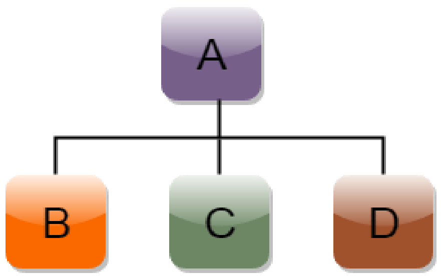
Multilevel inheritance is a process of deriving a class from another derived class.



Where A is a base class which inherited into class B, class B is inherited into class C.

[C]Hierarchical Inheritance

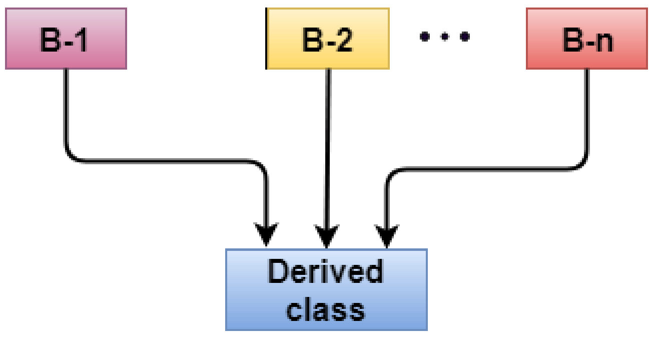
Hierarchical inheritance is defined as the process of deriving more than one class from a base class.



Where class B,C,and D derived from class A.

[D]Multiple Inheritance

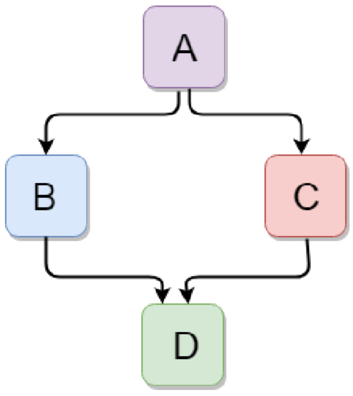
A derived class in inherited from more than one base class is called multiple inheritance.



Where Derived class is derived from B-1,B-2..B-n.

[E]Hybrid Inheritance

Hybrid inheritance is a combination of more than one type of inheritance.



Where class A is derived into class B and C, Class D is derived from class B and Class C.

[4]Encapsulation

- Encapsulation is a process of combining data members and functions in a single unit called class.
- This is to prevent the access to the data directly, the access to them is provided through the functions of the class.
- It is one of the popular features of Object Oriented Programming (OOPs) that helps in **data hiding**.

We can achieve encapsulation as per below

1. Make the entire data members private.
2. Create public get and put functions for each data member in such a way that the get function get the value of data member and put function put the value of data member.

Example:

```
Class demo
{
        int a,b;
        Public:
                Void get()
                {
                        Cout<<"Enter value of a:";
                        Cin>>a;
                        Cout<<"Enter value of b:";
                        Cin>>b;
                }

                Void put()
                {
                        Cout<<"value of a :"<<a;
                        Cout<<"value of b:"<<b;
                }

};

Void main()
{
        demo d;
        d.get();
        d.put();
        getch();

}
```

[5]Abstraction

- Wrapping the data into single unit without worried about its background process is known as Abstraction.
- Abstraction means displaying only essential information and hiding the details.
- Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

- Consider a real life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car.

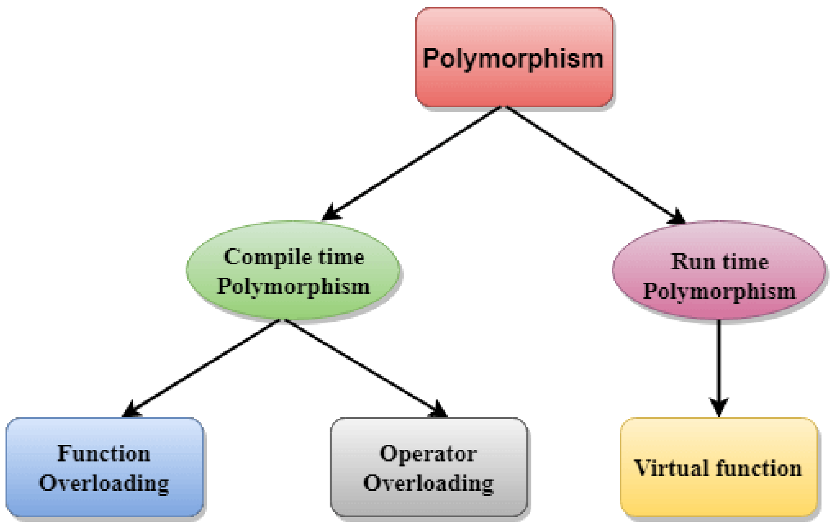
Example:

```
Class demo
{
        int a,b;
        Public:
                Void get()
                {
                        Cout<<"Enter value of a:";
                        Cin>>a;
                        Cout<<"Enter value of b:";
                        Cin>>b;
                }

        Void put()
        {
                        Cout<<"value of a :"<<a;
                        Cout<<"value of b:"<<b;
        }

};

Void main()
{

        demo d;
        d.get();
        d.put();
        getch();

}
```

As per above example public members and member functions are accessible through object but private cannot, how they cannot accessible or how public members and functions are accessible they are background process which does not need to know.

[6]Polymorphism

- The term "Polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a Greek word.
- Same name but different task is also known as polymorphism.
- Let's consider a real-life example of polymorphism. A lady behaves like a teacher in a classroom, mother or daughter in a home and customer in a market. Here, a single person is behaving differently according to the situations.
- There are two types of polymorphism



Compile time polymorphism

- A polymorphism is done at the time of compilation is called compile time polymorphism.
- There are two types of compile time polymorphism are there.

[A]function overloading

- Defining more than one function with same name but different arguments is called function overloading.

  Example

  Void sum();

  Void sum(int);

[B]operator overloading

- An operator which performs more than one task is called operator overloading.
- For example: "+" is used for addition of numbers and concatenation of string.

<u>Run time polymorphism</u>

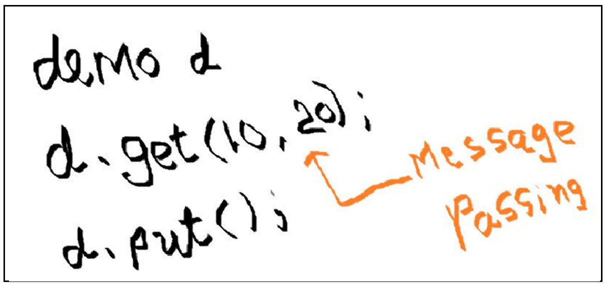
- A polymorphism done at the time of execution of program is called run time polymorphism.

- Method overriding or virtual function is example of run time polymorphism.

- It achieved in inheritance only.

[7]Message passing

- Objects communicate with one another by sending and receiving information to each other.
- A message for an object is a request for 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.



```
demo d
d.get(10,20);
d.put();
```

Que: 3 what is function explain member function, inline function and friend function.

- Function is a group of instructions which is used to perform specific task.
- Function is used for repetitive task.
- It is execute when it is called.

There are following types of functions in c++.

1. Member function
2. Inline function
3. Friend function

[1]Member function

- A function declares inside the class while its definition is defined inside or outside the class in known as member function.
- Definition of function is writing inside the class then declaration of function is not required while definition of function is written in outside the class then declaration of function is required.
- If defining the member function outside the class then Scope resolution operator (::) is required.
- At least one member function is defined into the public access specifier.
- It is called using object and dot(.) operator.

Example:

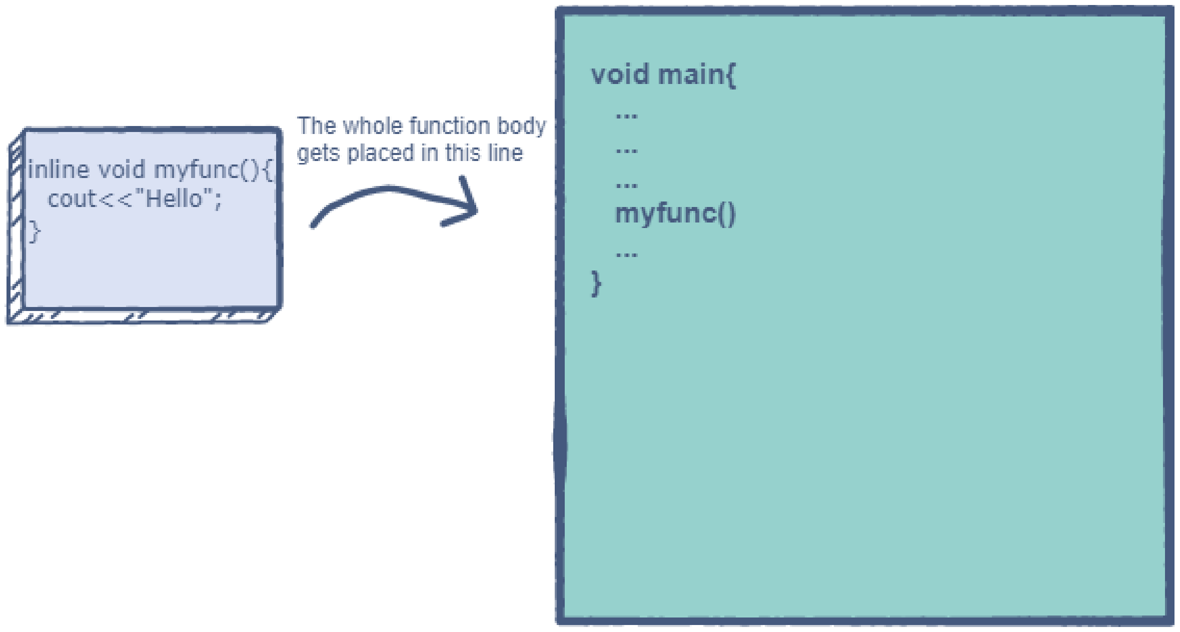
Definition inside the class	**Definition outside the class**
class student	class student
{	{
int roll;	int roll;
public:	public:
void get_data()	void get_data();
{	void put_data();
cin>>roll;	};
}	void student::get_data()
void put_data()	{
{	cin>>roll;
cout<<”Roll:”<<roll;	}
}	void student::put_data()
};	{
void main()	cout<<”Roll:”<<roll;
{	}
student s;	void main()
s.get();	{
s.put();	student s;
}	s.get();
	s.put();
	}

[2]Inline function

Normally, a **function call** transfers the control from the calling program to the called function. This process of jumping to the function, saving registers, passing a value to the argument, and returning the value to the calling function takes extra time and memory.

An **inline function** is a function that is expanded inline when it is invoked, thus saving time. The compiler replaces the function call with the corresponding function code, which reduces the overhead of function calls.

When an **inline function** is called, the compiler replaces all the calling statements with the function definition at run-time. Every time an inline function is called, the compiler generates a copy of the function's code, in place, to avoid the function call.



Rules:

Remember, inlining is only a request to the compiler, not a command. Compiler can ignore the request for inlining. Compiler may not perform inlining in such circumstances like:
1)If a function contains a loop. (for, while, do-while)
2) If a function contains static variables.
3) If a function is recursive.
4) If functions return type and the return statement doesn't exist in function body.
5) If a function contains switch or goto statement.
6) inline keyword is required when defining function definition outside the class.

<u>Advantages</u>

- In the inline function, we do not need to call a function, so it does not cause any overhead.
- It also saves the overhead of the return statement from a function.
- It does not require any stack on which we can push or pop the variables as it does not perform any function calling.

- An inline function is mainly beneficial for the embedded systems as it yields less code than a normal function.
- By default it is inline when it is define inside the class, if rules are break then it behaves like a normal function.

<u>Disadvantages</u>

- Due to code expansion, the size of the binary executable program is increased.
- Any change in the inline function code would need you to recompile the program to ensure it is updated.
- An increase in the page fault leads to poor program performance due to its increased executable size.
- For the systems with a large executable code, the call and return overhead time of its functions is negligible compared to their whole execution time. But, inlining the functions of such a program can bring down the speed of the system. So, it might not turn out to be a good option always, like in the case of large embedded systems which prefer high speed over sizeable executable size.

Example:

```
class A
{
 public:
    int max(int a, int b);//Inline function declaration
};
inline int A::max(int a, int b)//Inline function definition
{
          return (a>b)?a:b;
}
```

```
 class A
{
 public:
     void max(int a, int b)//Inline function declaration with definition
    {
          If(a>b)
                    Cout<<"a is max";
          else
                    cout<<"b is max";
    }
};
```

[3] Friend function

A function which has rights to access the private or protected data member of another class or outside the class is called friend function.

Properties of friend function.

- It should be define with friend keyword.
- It can be define with any access specifier either public or private or protected.
- Function declaration must be there where we want to make friend function between those classes.
- Function defition should be line as normal function, but it must have arguments and it should be as object.
- It can be invoked like a normal function without using the object, means it does not requires object and dot(.) operator to call this function.
- Members of class must be call using object and dot(.) operator.

Syntax:

```
Friend return_type function_name(argument_list);
```

Example :

[1] Friend function with a single class

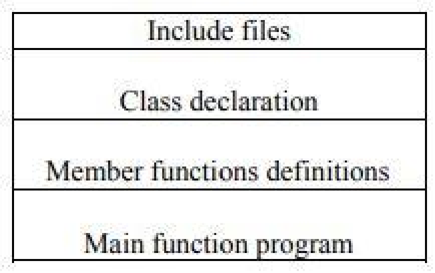
```
class demo

{

        int x,y;
        friend void max(demo);
        public:
                demo()
                {
                        x=10,y=20;
                }
} ;
void max(demo d)
{
        If(d.x > d.y)
        {
                cout<<"max is :"<<d.x;
        }
        else
        {
                cout<<"Max is :"<<d.y;
        }
}
void main()
{
        demo d;
        max(d);
        getch();
}
```

[2] Friend function with a two classes

```
class two;//early declaration
class one
{
        int x;
        friend void max(one,two);
        public:
                one()
                {
                        x=10;
                }
};
class two
{
        int y;
        friend void max(one,two);
        public:
               two()
                {
                        y=20;
                }
};
void max(one a,two b)
{
        if(a.x > b.y)
                cout<<"x is greater:"<<a.x;
        else
                cout<<"y is greater:"<<b.y;
}
void main()
{
        one A;
        two B;
        max(A,B);
        getch();
}
```

Que: 4 write a note on: Structure of 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, are as follow.



[1]include files

- Similar to C, C++ programs are also dependent on certain header files.
- Each header file has a '. h' extension.
- This file is pre-compiled, it means it does not required to recompile it, it required only to link it.
- For example:
  - #include<iostream.h>
  - #include<conio.h>
  - #include<string.h>

[2] class declaration

- Class is a collection of data members and member functions.
- It consist function declaration or function definition.
- Members and member function define inside the specific access specifier such as public, private and protected.
- Every member separated by comma (,).
- Default access specifier is private.
- The class declaration must be enclosed with curly braces ({ }) and terminated by a semicolon (;).

Syntax	Example

class <class name>
{
Variables declaration; // data members
Function declaration/definition; // member functions
}; // End of class
	class student
	{
	int roll;
	char name[20];
	Public:
	void get();
	void put();
	};

[3]Member function definition:

- A member function definition can be done outside or inside the class declaration.
- If the function is small then define it inside the class.
- When the function is large, define it outside the class.
- In this case, the prototype or declaration of a function should be reported within the class.
- If definition of member function define outside the class then name of the class and scope resolution operator (::) is required in definition of member function.

Example

```
void student::get()
{
        cout<<"Enter roll no:";
        cin>>roll;

        cout<<"Enter name:";
        cin>>name;
}
void student::put()
{       cout<<"roll no is:"<<roll<<endl;
        cout<<"Name is:"<<name;
}
```

[4] main function program

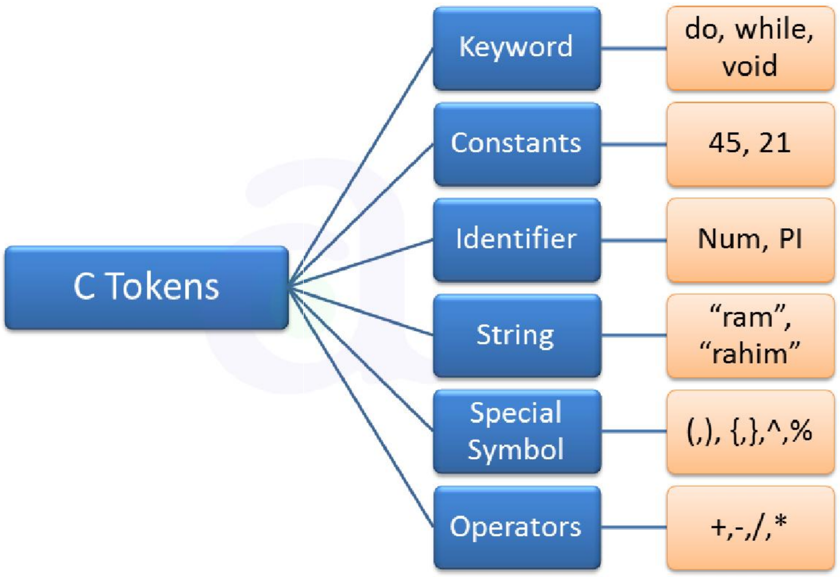
- Main function is a heart of c/c++ program.
- Execution of every program starts with here.
- All C++ statements that need to be executed are written within main ( ).
- The compiler executes all the instructions written within the opening and closing curly braces' {}' that enclose the body of main ( ).
- 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.

Example

```
void main()
{
        student s;
        s.get();
        s.put();
}
```

Que : 5 what is Token? Explain types of token in c++

A token is the smallest unit of a program that the compiler understands
In C++, Tokens are divided into



1. Keywords

In a programming language, there are certain reserved words that have fixed meaning. These words are known as keywords. We cannot use a keyword for purposes other than what it's reserved for.

There are 32 keywords supported in C language and additional keyword.

auto	double	int	struct	class	protected
break	else	long	switch	inline	this
case	enum	register	typedef	operator	virtual
char	extern	return	union	delete	
const	float	short	unsigned	friend	
continue	for	signed	void	new	
default	goto	sizeof	volatile	private	
do	if	static	while	public	

2. Identifiers

- We can give names to various elements of a program like variables, functions, structures, etc. These user-defined names are called identifiers.
- Identifiers must be unique as we use them in the program execution.
- We need to follow some rules for naming identifiers:

1. An identifier name should begin with a letter or an underscore (_).
2. An identifier name should be made up of letters, digits and underscores only.
3. Special characters and white spaces are not allowed.
4. No keyword can be used as an identifier.
5. Identifier names are case sensitive.
6. Length of an identifier name should not exceed 31 characters, beyond which it becomes insignificant.

Examples of some valid identifiers are: Name, age, add_numbers, _students, etc.
Examples of some invalid identifiers are: Name@, 6age, new, etc.

3. Constants

Constants are expressions whose values remain fixed. Once defined, we cannot change the value of a constant. These can also be referred to as literals.

Constants can be of integer, floating-point, character, string data types.

For example,
const float pi = 3.14;
Here, the value of constant pi cannot be changed in the entire program.

4. Strings

A string stores a sequence of characters. It terminates with a null character '\0'. Unlike characters, strings in C++ are always enclosed within double quotes (" ").

Example – char name[ ] = "TechVidvan";

5. Special Symbols

There are some special symbols in C++ that have special meaning to the compiler. We cannot alter their meaning. List of special symbols in C++ are:

Special Symbol	Name
[ ]	Square Brackets
()	Parentheses
{ }	Curly braces
,	Comma
:	Colon
;	Semicolon
*	Asterisk
#	Hash/ Preprocessor
.	Dot
~	Tilde

6. Operators in C++

- Operators are symbols that operate on operands.
- These operands can be variables or values.
- Operators help us to perform mathematical and logical computations.
- Operators in C++ are classified into following types based on the number of operands they operate on:

Unary Operators:

These act upon one operand. For example, increment operator (++).

Binary Operators:

They operate on two operands. For example, addition operator (+).

Ternary Operator:

There is a ternary operator in C++ that acts on three operands. It is the ?: conditional operator.

On the basis of nature of operation, operators in C++ are classified into following six types:

- Arithmetic(+,-,*,/,%)
- Assignment(+=,-=,*=,/=,%=)
- Relational(>,<.>=,<=,==,!=)
- Logical(&&,||,!)
- Bitwise(&,|,~,<<,>>,^)
- Other Operators(,.::)

Que : 6 What is Operator? Explain types of Operators available in c++.

- Operator is a special symbol which is used to perform a specific task.
- There are main three types of operator such as
  - Unary
  - Binary
  - Ternary

C programming support various operator such as

- Arithmetic operator
- Assignment operator
- Relational operator
- Logical operator
- Bitwise operator
- Increment or decrement operator
- Conditional operator

C++ support other operators like,

[1] Scope resolution operator (::)
[2] Memory management operator (new, delete)
[3] Manipulators (endl,setw)
[4] Member dereferencing operator (::* , * , ->*)

[1] scope resolution operator

The scope resolution operator is used to reference the global variable or member function that is out of scope.

Therefore, we use the scope resolution operator to access the hidden variable or function of a program.

The operator is represented as the double colon (::) symbol.

Uses of the scope resolution Operator

1. It is used to access the hidden variables or member functions of a program.
2. It defines the member function outside of the class using the scope resolution.
3. It is used to access the static variable and static function of a class.
4. The scope resolution operator is used to override function in the Inheritance.

[1] It is used to access the hidden variables or member functions of a program.

We can access global variable or global function which is already define inside the class.

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

int x=10;
void check()
{
        cout<<"Global function"<<endl;
}
class demo
{
        int x;
        public:
                demo()
                {
                        x=20;
                }
                void check()
                {
                        cout<<"inside class"<<endl;
                        cout<<"Inside member:"<<x<<endl;
                        cout<<"Outside member::"<<::x<<endl;
                }
                void process()
                {
                        check();
                        ::check();
                }
};

void main()
{
        clrscr();
        demo d;
        d.process();
        getch();
}
```

[2] It defines the member function outside of the class using the scope resolution.

We can use :: operator when member function define outside the class.

```
class demo
{

        public:
                void msg();

};

void demo::msg()
{
        Cout<<"hello";
}

void main()
{
        clrscr();
        demo d;
        d.msg();
        getch();
}
```

[2] Memory management operator (new, delete)

Memory management is a process of managing computer memory, assigning the memory space to the programs to improve the overall system performance.

In C language, we use the **malloc()** or **calloc()** functions to allocate the memory dynamically at run time, and free() function is used to deallocate the dynamically allocated memory. C++ also supports these functions, but C++ also defines unary operators such as **new** and **delete** to perform the same tasks, i.e., allocating and freeing the memory.

New operator

A **new** operator is used to create the object while a **delete** operator is used to delete the object. When the object is created by using the new operator, then the object will exist until we explicitly use the delete operator to delete the object. Therefore, we can say that the lifetime of the object is not related to the block structure of the program.

Syntax

pointer_variable = **new** data-type

Example 1:

int *p;
p = new int;
	float *q;
	q=new float;

Example 2:

Assigning array size to array

Syntax:
Pointer-variable = **new** data-type[size];

int *a1 = new int[8];	float *b1=new float[8]

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, as shown below:

delete pointer_variable;

In the above statement, **'delete'** is the operator used to delete the existing object, and **'pointer_variable'** is the name of the pointer variable.

delete p;
delete q;

The dynamically allocated array can also be removed from the memory space by using the following syntax:

delete [size] pointer_variable;

In the above statement, we need to specify the size that defines the number of elements that are required to be freed. The drawback of this syntax is that we need to remember the size of the array. But, in recent versions of C++, we do not need to mention the size as follows:

delete [ ] pointer_variable;

[3] Manipulators (endl,setw)

Manipulators are operators used in C++ for **formatting output**. The data is manipulated by the programmer's choice of display.

endl

This manipulator has the same functionality as the 'n' newline character.

For example:

```
1. cout << "Exforsys" << endl;
2. cout << "Training";
```

setw

This manipulator sets the minimum field width on output.

```
Syntax:
setw(x)
```

Int x=123;

cout<<setw(10)<<x;

							1	2	3

[4]Member dereferencing operator (::* , * , ->*)

- *In computer programming, a* ***dereference operator,*** *also known as an indirection operator, operates on a pointer variable.*
- It returns the location value, or **l-value** in memory pointed to by the variable's value. In the C programming language, the **dereference operator** is denoted with an asterisk **(*)**.

For example, in C, we can declare a variable **M** that holds an integer value, and a variable ***ptr** that holds a pointer to an integer value in memory:

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

          void main()
          {
                    int M;
                    int *ptr;
                    M = 5;
                    ptr= &M;
                    cout << *ptr_p;
          }
```

In C++ Programming we have three different dereferencing operators.

1. Pointer to a member declarator (::*)
2. Pointer to member operator (.*)
3. Pointer to member operator (->*)

1. Pointer to a member declarator (::*)

This operator is used for declaring a pointer to the member of the class. It is possible to take the address of a member of a class and assign it to a pointer. The address of a member can be obtained by applying the operator & to a "fully qualified" class member name.

A class member pointer can be declared using the operator**:: *** with the class name. We can define a pointer to the member m of class ABC as follows:

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

class demo
{
        public:
        int j;

        demo()
        {
                j=20;

        }
};
void main()
{
        clrscr();
        int demo::*p=&demo::j;  //pointer to member;
        demo d;

        cout<<"Value of j:"<<"d.j<<endl;
        cout<<"Value of pointer member:"<<d.*p;

        getch();
}
```

Output:

```
Value of j:20
Value of pointer member:20
```

2. **Pointer to member operator(.*)**
This operator is used to access value of member using pointer.

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

class demo
{
        public:
        int j;

        demo()
        {
                j=20;

        }
};
void main()
{
        clrscr();
        int demo::*p=&demo::j;  //pointer to member;
        demo d;

        cout<<"Value of j:"<<"d.j<<endl;
        cout<<"Value of pointer member:"<<d.*p; //access value of member using
pointer

        getch();
}
```

Output:

```
Value of j:20
Value of pointer member:20
```

3. object Pointer to member operator (->*)

This operator is used to point to a member using pointer of object.

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

class demo
{
        public:
        int j;

        demo()
        {
                j=20;

        }
};
void main()
{



        clrscr();
        int demo::*f=&demo::j;
        demo d,*x;
        x=&d;

        cout<<"Object pointer pointing to member:"<<x->*f;\\object pointer
pointing to member
        getch();
}
```

Output:

```
Object pointer pointing to member:20
```

Que : 7 Explain static member and member function with proper example.

In c++ programming memory of objects are allocated separately. Separate memory location is required for the member variables will holds different data values for different objects.

For static member it required single memory block for all objects of same class.

Static data member

Static is a keyword which is used to define data member as static.

A data member of a class can be qualified as static.

It has following important properties.

1) It is initialized when first object of class is created.
2) Default value is 0.
3) Only one copy is created for all objects of same class. Which share between all objects of same class?
4) It is visible only within the class but its lifetime is the entire program.
5) It cannot be access through the object and (.) operator as well as class_name and :: operator.
6) It can be define outside the class using name of class and :: operator because it has same memory location for all objects of same class.
7) It's a class variable so no we can access it without creating object of the class

Example

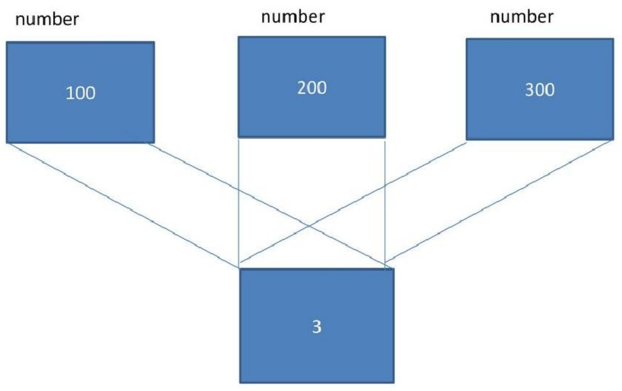
```
#include<iostream.h>
#include<conio.h>
class demo
{
        public:
                static int number;
                int x;
                demo()
                {
                        x=0;
                }
                void count()
                {
                        number+=100;
                        x++;
                }
```

```
                  void display()
                     {
                              Cout<<"x:"<<x<<endl;
                              cout<<number<<endl;
                     }
};

int demo::number; //definition of static variable

void main()
{
          Cout<<demo::number;
          demo d,d1,d2;
          clrscr();
          d.count();
          d1.count();
          d2.count();

          d2.display();
          d1.display();
          d.display();

          cout<<demo::number;
          getch();
}
```

Output:

```
Number:0

X:1
Number:300

X:1
Number:300

X:1
Number:300
```

Sharing a static Data Member



Count (common to all three objects

Static member function

- Like static member we can also define the static member function using static keyword.
- Static member function can have access only static data members or member functions.
- It can be called using class name and :: operator as well as object and dot (.) operator.
- It's a function of class so should be define in public access specifier otherwise it cannot be access outside the class.
- It's a function of class hence object is not required to access it, we can access it without creating object of the class using name of class and:: operator.

Example

```
#include<iostream.h>
#include<conio.h>
class demo
{
        public:
                static int number;
                int x;
                demo()
                {
                        x=0;
                }
                static void count()
                {
                        number+=100;
                        // x++; not permitted
                }
```

```
                  void display()
                    {
                              cout<<"x:"<<x<<endl;
                              cout<<"Number:"<<number<<endl;
                    }
};

int demo::number; //definition of static variable

void main()
{
          clrscr();
          demo::count();//call without object

          demo d1,d2;
          d1.count();
          d2.count()

          d1.display();
          d2.display();

          getch();
}
```

Output:

```
X:1
Number:200

X:1
Number: 300
```

Que: 7 Explain input output statements in c++

C++ I/O occurs in streams, which are sequences of bytes. If bytes flow from a device likes a keyboard, a disk drive, or a network connection etc. to main memory, this is called **input operation** and if bytes flow from main memory to a device likes a display screen, a printer, a disk drive, or a network connection, etc., this is called **output operation**.

[1]The Standard Output Stream (cout)

The predefined object **cout** is an instance of **ostream** class. The cout object is said to be "connected to" the standard output device, which usually is the display screen. It must required iostream.h header file.The **cout** is used in conjunction with the stream insertion operator, which is written as << which are two less than signs as shown in the following example.

cout<<"Hello world";

int x=10;

char *p="world";

cout<<x<<p;

The << operator is overloaded to output data items of built-in types integer, float, double, strings and pointer values.

The **insertion operator <<** may be used more than once in a single statement as shown above and **endl** is used to add a new-line at the end of the line.

[2]The Standard Input Stream (cin)

The predefined object **cin** is an instance of **istream** class. The cin object is said to be attached to the standard input device, which usually is the keyboard. It must required iostream.h header file. The **cin** is used in conjunction with the stream extraction operator, which is written as >> which are two greater than signs as shown in the following example.

Int x,y;

cin>>x<<y;

The C++ compiler also determines the data type of the entered value and selects the appropriate stream extraction operator to extract the value and store it in the given variables.

The stream **extraction operator >>** may be used more than once in a single statement. To request more than one datum you can use the following −

cin >> name >> age;

[3]Standard end line (endl)

The **endl** is a predefined object of **ostream** class. It is used to insert a new line characters and flushes the stream.

cout<<"Hello"<<endl;

cout<<"world";

output:

```
Hello

World
```