C++ UNIT - 2

Topic: Explain structure in detail.

- ✓ A structure combines logically related data items into a single unit.
- ✓ It is a user define data types.
- ✓ It can be used to create a variables, which can be used in the same way as variables of standard data types.

- ✓ **Syntax:**

```
struct structure name
{
     Data type member1;
     Data type member2;
     -----
};
```

- ✓ According to syntax the structure declaration starts with struct keyword with structure name.
- ✓ The data type of each variable is specified in the individual member declaration.
- ✓ The closing bracket is terminated with a semicolon.

Example:

- **Write a c++ program for enter the student roll no and name from user using structure.**

```
#include<iostream.h>
#include<conio.h>
struct std
{
      int no;
      char name[40];
      void input()
      {
            cout<<"Enter no:";
            cin>>no;
            cout<<"Enter name:";
            cin>>name;
      }
      void display()
      {
         cout<<endl<<"No is:"<<no;
         cout<<"\n Name is:"<<name;
      }
};

void main()
{
      clrscr();
      std s1;
      s1.input();
      s1.display();
      getch();
}
```

C++ Classes and Objects

Class:

A class in C++ is the building block, that leads to Object-Oriented programming. 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 C++ class is like a blueprint for an object.

For Example:

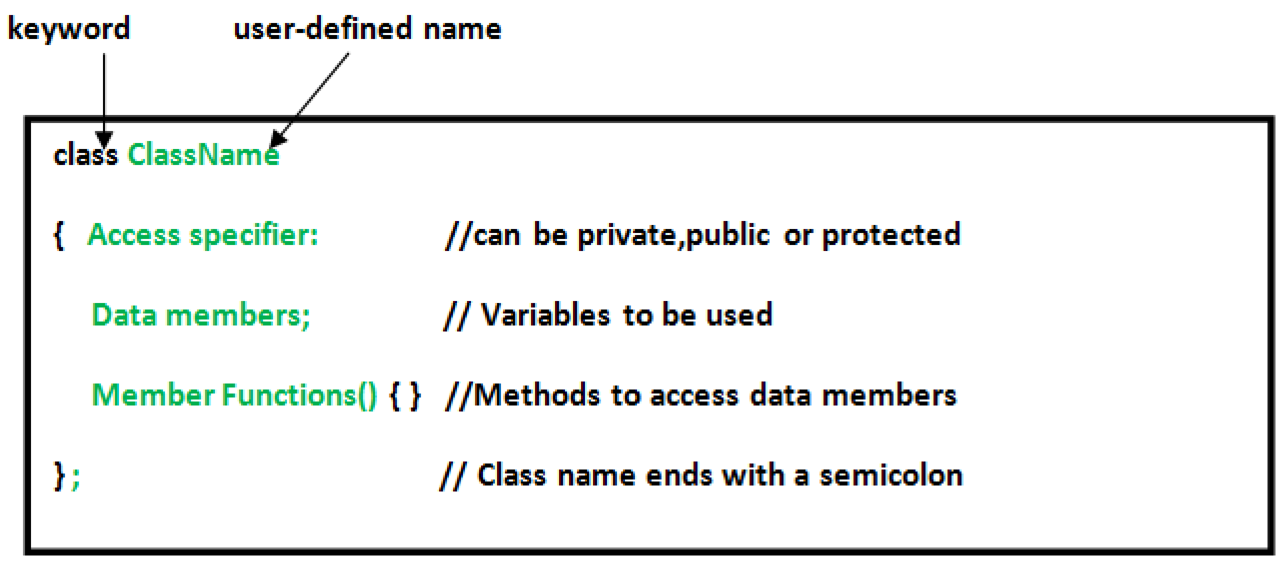
Consider the Class of **Cars**. There may be many cars with different names and brand but all of them will share some common properties like all of them will have *4 wheels*, *Speed Limit*, *Mileage range* etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

- A Class is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behaviour of the objects in a Class.
- In the above example of class *Car*, the data member will be *speed limit*, *mileage* etc and member functions can be *apply brakes*, *increase speed* etc.

An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

Defining Class and Declaring Objects

A class is defined in C++ using keyword class followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



```
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
```

Declaring Objects: When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.

Syntax:

```
ClassName ObjectName;
```

Accessing data members and member functions:

` The data members and member functions of class can be accessed using the dot('.') operator with the object. For example if the name of object is *obj* and you want to access the member function with the name *printName()* then you will have to write *obj.printName()* .

Accessing Data Members

The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member.

This access control is given by Access modifiers in C++. There are three access modifiers : **public, private and protected**.

};

Example :

```
#include<iostream.h>

#include<conio.h>

class Neeks

{

    // Access specifier

    public:

    // Data Members

    char neekname;

    // Member Functions()

    void printname()

    {

       cout << "Neekname is: " << neekname;

    }

};
```

```
int main()

{

   // Declare an object of class Neeks

    Neeks obj1;

    // accessing data member

    obj1.neekname = ‘a’;

    // accessing member function

    obj1.printname();

    return 0;

    getch();

}
```

Output:

```
Neekname is: a
```

Member Functions in Classes

There are 2 ways to define a member function:

- Inside class definition
- Outside class definition

To define a member function outside the class definition we have to use the scope resolution :: operator along with class name and function name.

```
#include<iostream.h>

#include<conio.h>

class Neeks

{

    public:

    char neekname;

    int id;

    void printname();

     void printid()

     {

        cout << "Neek id is: " << id;

    }
};

void Neeks::printname()

{

    cout << "Neekname is: " << neekname;

}

int main()

{

    Neeks obj1;

    obj1.neekname = ‘a’;

    obj1.id=15;

    obj1.printname();

    cout << endl;

    obj1.printid();

    getch();

}
```

Output:

```
Neekname is: a

Neek id is: 15
```

.

- **Write a c++ program for member function with arguments.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
class data
{
	int no;
	char name[20];
	public:
	void input(int,char *);
	void output();
};
void data::input(int n,char nm[])
{
	no=n;
	strcpy(name,nm);
}

void data::output()
{
	cout<<"\n number is:"<<no;
	cout<<"\n Name is :"<<name;
}
```

```
int main()
{
	int no;
	char name[20];
	data d1;
	clrscr();
	cout<<"Enter the number: ";
	cin>>no;
	cout<<"Enter the Name: ";
	gets(name);
	d1.input(no,name);
	d1.output();
	getch();
	return 0;
}
```

➢ **Write a c++ program for nesting of member function.**

```
#include<iostream.h>
#include<conio.h>
class number
{
	int no1,no2;
	public:
	void input(); void
	output(); int
	add();
};
void number::input()
{
	cout<<"\n Enter any two Numbers: ";
	cin>>no1>>no2;
}
int number::add()
{
	return(no1+no2);
}

void number::output()
{
	cout<<endl<<"No1 is: "<<no1; cout<<endl<<"No2 is:
	"<<no2; cout<<endl<<"Addition of two number is:
	"<<add();
}
```

```
void main()
{
	clrscr();
	number n1;
	 n1.input();
	n1.output();
	getch();
}
```

➤ **Write a c++ program from access the private member function.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class std
{
	int no;
	char name[50];
	void input()
	{
		cout<<"\n Enter number:";
		cin>>no;
	}
	    public:
	    void display();
}s1;



void std::display()
{
	input();
	cout<<"\n Square of number is: "<<no*no;
}


void main()
{
	clrscr();
	std s2;
	s1.display();
	s2.display();
	getch();
}
```

Topic: How to make outside function inline?

- ✓ An inline function is working as a macro, any call to this function in program is replaced by the function calling itself.
- ✓ In c++ all the member functions that are defined within the class specification is inline by default.
- ✓ member function declare outside the class specification can be made inline by prefixing inline keyword to its definition.

➢ Write a c++ program for inline outside member function.

```
#include<iostream.h>
#include<conio.h>
class number
{
        int no1,no2;
        public:
        void input(int,int);
        void output();
};
inline void number::input(int n1,int n2)
{
        no1=n1;
        no2=n2;
}
inline void number::output()
{
        cout<<endl<<"No1 is: "<<no1;
        cout<<endl<<"No2 is: "<<no2;
}
void main()
{
        clrscr();
        number n1,n2;
        n1.input(10,20);
        n2.input(20,30);
        cout<<"The Numbers are: ";
        n1.output();
        n2.output();
        cout<<endl<<"Size  of  operator  1  is:  "<<sizeof(n1);
        cout<<endl<<"Size  of  operator  2  is:  "<<sizeof(n2);
        getch();
}
```

Topic: Explain array within a class.

✓ An array is derived data type. Array is a group of related data items of same type which share a common name. The array can also be used as a member variable.

➢ **Write a c++ program for array with in a class.**

```
#include<iostream.h>

#include<conio.h>

class sum

{

            int no[10];

             int s;

             public:

            void input();

            void output();

};

void sum::input()

{

            for(int i=0;i<10;i++)

            {

                        cout<<"\n Enter the value of no ["<<i+1<<"]: ";

                        cin>>no[i];

            }

}
```

```
void sum::output()
{
        int tot=0;
        for(int i=0;i<10;i++)
        {
                tot=tot+no[i];
        }
        cout<<"\n Total of all Number is: "<<tot;
}
void main()
{
        sum s1;
        clrscr();
         s1.input();
        s1.output();
        getch();

}
```

Topic: Explain Memory allocation for objects.

- ✓ The object is a variable of class. A class contain both member variable and member function. An object occupies number of bytes in a memory.
- ✓ The member function are created and placed in memory only one when, they are defined in the class declaration because all the objects belonging to that class use the same member functions. When the objects are created.
- ✓ The data members are placed in memory when each object is defined because all object have the separate data member and allocate separate memory of each data member for each object.
  - ✓ The sizeof operator is used to find out size of standard data type.
  - ✓ The size of any class object can also be find using it.

Topic: Explain Static data Member.

- ✓ Sometimes situation occurs when need one or more common data member, which are accessible to all the objects of the class.

  - ✓ **Syntax:**

```
Class Class_Name
{
Static data_type  Data_member;
};
```

- ✓ According to syntax, the static data member can be created by static keyword .
- ✓ It is initialized during its definition outside all the member functions.
- ✓ The static data member is stored separately rather than as a part of an object.
- ✓ it is associated with the class itself, rather than with any class object.
- ✓ It is also known as **class variable.**
- ✓ Like:

```
Datatype class_name :: Datamember= value;
```

- ✓ Consider following characteristics of static data member as follows:

1. Only one copy of that member is created for the entire class and is shared by all objects of that class . Al l the changes made to that member at same memory location. So when display the value of that always get last value.
2. When first object of class is created it is initialized to zero bydefault.
3. It is visible only within the class, but its life is entire the program.

- **Write a c++ program for static data member.**

```
#include<iostream.h>
#include<conio.h>
class test
{
	static int count;
	int num;
	public:
	void setdata(int n)
	{
		num=n;
		++count;
	}
	void display()
	{
		cout<<endl<<"Value of number: "<<num;
		cout<<endl<<"Value of static variable: "<<count;
	}
};
int test::count;
void main()
{
	clrscr();
	test t1,t2,t3;
	t1.display();
	t1.setdata(10);
	t1.display();
	t2.setdata(20);
	t2.display();
	t3.setdata(30);
	t3.display();
	getch();
}
```

Topic: Explain Static member Function.

- ✓ Same as static member variable, class can also have static member function.
- ✓ Consider following points about static member function.

1. Only one copy of static member function exists for all instance of class.
2. A static member function can access only static data member.
3. A static member function can be called using the class_name instead of its object. Such as:

Class_name :: Function_name

➢ **Write a c++ program for static member function.**

```
#include<iostream.h>

#include<conio.h>

class test

{

        static int count;

        int num;

        public:

        void setdata(int n)

        {
                num=n;

                ++count;

        }

        static void display()

        {

                cout<<endl<<"Value of static variable: "<<count;

        }

        void dispnum()

        {

                cout<<endl<<"Value of number is: "<<num;

        }

};
```

```
int test::count;

void main()

{

        clrscr();

        test t1,t2,t3;

        test::display();

        t1.setdata(10);

        test::display();

        t1.dispnum();

        t2.setdata(20);

        test::display();

        t2.dispnum();

        t3.setdata(30);

        test::display();

        t3.dispnum();

        getch();

}
```

Topic: Explain array of objects.

- ✓ An array is a group of same data type and stored in adjacent memory location.
- ✓ As an array of any other standard data type, the array of class object can also be created.
- ✓ It is known as an array of objects.
- ✓ Consider the following format to create array of object: Class class_name

```
{

Private:

    Variable declaration; Function declaration;

Public:

    Variable declaration; Function declaration;

};


Class_name  object[size];
```

- ✓ Here, size defines the size of the array of class object.

➢ **Write a c++ program for static member function.**

```
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
class std
{
        int rlno;
        char stnm[30];
        public:
        void input()
        {
                cout<<"Enter rono:";
                cin>>rlno;
                cout<<"Enter Std name:";
                gets(stnm);
        }
        void output()
        {
                cout<<"\n Rollno is: "<<rlno;
                cout<<"\n Name is: "<<stnm;
        }
};
```

```
void main()
{
        clrscr();
        std s[3];
        for(int i=0;i<3;i++)
        {
                s[i].input();
        }
        for(i=0;i<3;i++)
        {
                s[i].output();
        }
        getch();
}
```

Topic: Explain object as function argument.

- ✓ Like any other data type, an object can be passed as an argument to a function.
- ✓ This can be achieved by following ways:

1. Pass – by – value
2. Pass - by – reference
3. Pass – by – pointer

1. Pass – by – value

- ✓ In the case of pass – by – value a copy of object is passed to the function.
- ✓ It modification made to the object inside function is not reflected to the object that is used to call the function.

➢ **Write a c++ program for passing the object by value.**

```
#include<iostream.h>

#include<conio.h>

class calc

{

        int no1,no2;

        public:

        void input(int n1,int n2)

        {

                no1=n1;

                no2=n2;

        }

        void output()

        {

                cout<<endl<<"No1= "<<no1;

                cout<<endl<<"No2= "<<no2;

        }
```

```
	void add(calc c1,calc c2)

	{

		no1=c1.no1+c2.no1;

		no2=c1.no2+c2.no2;

	}

};

int main()

{

	clrscr();
	calc c1,c2,c3;

	c1.input(5,10);

	c2.input(20,30);

	c1.output();

	c2.output();

	c3.add(c1,c2);

	cout<<endl<<"Addition of two object is: ";

	c3.output();

	getch();

	return 0;

}
```

2. **Pass – by – reference**

- ✓ In the case of pass – by – reference any changes made to the object inside the function is reflected in the actual object.
- ✓ Here, object is passed same as pass by value but received in the function body with its reference.

3. **Pass – by – pointer**

  - ✓ In the case of pass – by – pointer an address of object passed to function.
  - ✓ Any changes made in value of object function it will also affect the original value.
  - ✓ The member of object passed by pointer is accessed using the operator.

Topic: Explain returning object from function.

✓ Similar to sending objects as arguments to the function, it is also possible to return object from the function.

➢ **Write a c++ program for returning object from function.**

```
#include<iostream.h>
#include<conio.h>
class a
{
        int x,y,z;
        public:
        void input()
        {
                cout<<endl<<"Enter Three Value..";
                cin>>x>>y>>z;
        }
        void output()
        {
                cout<<endl<<"x: "<<x<<"\ty:  "<<y<<"\tz:  "<<z;
        }
a add(a a1)
{
        a a2; a2.x=a1.x+a1.y+a1.z;
        return a2;
}
};


void main()
{
        a a1,a2;
        clrscr();
        a1.input();
        a1.output();
        a2=a2.add(a1);
        a2.output();
        getch();
}
```

Topic: Explain friend function.

✓ As the object oriented concept the private member of class can not be accessed from outside the class.

✓ However, under certain situations, sometimes requirement to use a common function with two classes.

✓ In c++ this is achieved by using the concept of **friends.** It permits a function of another class to access different class's private members.

✓ The friendly function must be prefixed by keyword **friend**.

✓ The function calling is same as ordinary function without scope of class.

✓ A function calling is same as normal function without the dot(.) operator.

✓ The special characteristics of a friend function are as follows:

1. It is not declared as a member of any class. So the scope of friend function is not limited to the class in which it has been declared as a friend.
2. It can only be invoked like a normal function without the help of any object because it is not in the scope of class.
3. It can access the private members of a class using object name and dot(.) operator.
4. It can be declared either in the public or private section without affecting its accessibility.
5. It has the objects as arguments.

✓ Format of friend function.

```
Class sample
{
        Int x,y;
        Public:
        friend void add();//declaration of friend function
};

//definition of friend function like normal function

Void add()

{

               //body of function;

}

int main()

{
        sample obj;
        add(obj);
}
```

➢ **Write a c++ program for friend function.**

```
#include<iostream.h>
#include<conio.h>
class data2;
class data1
{
	int x;
	public:
	void input()
	{
		cout<<endl<<"Enter Value of x:";
		cin>>x;
	}
	void output()
	{
		cout<<endl<<"Value of x is: "<<x;
	}
	friend int max(data1,data2);
};
class data2
{
	int y;
	public:
	void input()
	{
		cout<<endl<<"Enter value of y:";
		cin>>y;
	}
	void output()
	{
		cout<<endl<<"Value of y is: "<<y;
	}
	friend int max(data1,data2);
};
int max(data1 d1,data2 d2)
{
	if(d1.x>d2.y)
	{
		return (d1.x);
	}
	else
	{
		return (d2.y);
	}
}
```

```
int main()
{
	clrscr();
	data1 d1;
	data2 d2;
	d1.input();
	 d2.input();
	d1.output();
	d2.output();
	cout<<endl<<"Maximum value is: "<<max(d1,d2);
	getch();
	return(0);
}
```

Topic: Explain const member function.

- ✓ If the member function of a class access the class data member without modifying them, then declare such function as **const** (constant) function.
  - ✓ syntax :

  return type function name(argument)const.
- ✓ According to above syntax the keyword **const** is placed at the end of function header, at the time of declaring its prototype and its definition.
  - ✓ The compiler will generate an error message if function try to change the data values.

➢ **Write a c++ program for const member function**.

```
#include<iostream.h>
#include<conio.h>
class data
{
        int i;
        public:
        void set(int x)
        {
                i=x;
        }
        void change()const
        {
                cout<<endl<<"you can not change the value of const function:";
        }
        void show()
        {
                cout<<endl<<"i: "<<i;
        }
};

void main()
{
        clrscr(); data
        d; d.set(10);
        d.show();
        d.change();
        d.show();
        getch();
}
```

Topic: Explain pointer to member function.

- ✓ pointer variable holds the address of another variable.
- ✓ It is also possible to take the address of member of a class and assign it to a pointer.
- ✓ The address of a member can be obtained with using the address of operator (&) to a class member name.
- ✓ C++ provides special operator pointer to member (:: *) with the class_name to declare a class member pointer.
- ✓ **For ex:**

```
class sample
{
        int x;
        public:
        Void display();
};
```

✓ Pointer to member can be created as follows:

int sample :: *p=&sample :: x;

✓ Here, the p is pointer to member and the portion **sample :: *** means **"pointer-to-member of sample class"** and the portion **&sample :: x** means **"address of the x member of class sample".**

- ✓ int *p=&x;
- ✓ here, this statement is invalid.
- ✓ It indicate pointer of normal variable.
- ✓ The x is not only int variable but a member variable of class sample.
- ✓ The scope resolution operator must be attached to both the pointer and the member.
- ✓ A pointer can be used to access the member variable x is as follow.

```
sample s;
cout<<s.*p;        //display value of x cout<<s.x;

//display value of x
```

- ✓ The dereferencing operator(->*) is used to access a member when use pointer to both object and member.
- ✓ The dereferencing operator(.*) is used when the object use only member pointer.

✓ For ex: sample *s;

```
s=&x; cout<<s->*p;
cout<<s->x;
```

✓ The pointer member function can be invoked using the dereferencing operators in the main() as follows:

(object name.*pointer_to_member function)(value); (pointer-to-object ->*pointer-to-member function)(value);

➢ **Wrire a c++ program for pointer to member function**.

```
#include<iostream.h>
#include<conio.h>
class data
{
        int a,b;
        public:
        void setdata(int x,int y)
        {
        a=x;
        b=y;
        }
        friend int add(data);
};

int add(data d1)
{
        //*p1=&a;
        int data::*p1=&data::a;
        int data::*p2=&data::b;
        //data d1;
        data *tmp=&d1;
        int sum;
        //sum=d1.a+d1.b;
        //sum=d1.*p1+d1.*p2;
        //sum=tmp->*p1+tmp->*p2;
        //sum=tmp->a+tmp->b;
        sum=d1.*p1+tmp->*p2;
        return(sum);
}

void main()
{
        clrscr();
        data d;
        d.setdata(5,10);
        cout<<endl<<"Addition of two number is: "<<add(d);
        getch();
}
```

Topic: Explain nested class and local class.

- ✓ When specifies any class inside another class, then its known as nested class.
- ✓ For ex:

```
class outer
{
        class inner
        {
                -------
                -------



        };
        inner obj;
        public:
        --------
        --------
};
```

- ✓ As shown in above ex the class outer has one class inside it called inner.
- ✓ It will create an object of class inner and all member function of class outer can access member variable of inner class by its object.
- ✓ It is also possible to access the class outside the other class, but then use the inner class with the scope of outer such as outer :: inner.
- ✓ When classes defined and used inside function block then it is known as local class.

- ✓ Here, the function sample can has a class_name inner local class can use global.
- ✓ Variables and static variable declared inside the function but cannot use automatic local variables.
- ✓ The function demo can not access the private member of a local class.

➢ **Write a c++ program for nested class.**

```
#include<iostream.h>
#include<conio.h>
class A
{
	public:
	class B
	{
		privat:
		int
		num;
		public:

	void getdata(int n)
	{
		num = n;
	}
	void putdata()
	{
		cout<<"The number is "<<num;
	}
	};
};


int main()
{
	clrscr();
	cout<<"Nested classes in C++"<< endl;
	A :: B obj;
	obj.getdata(9);
	obj.putdata();
	getch();
	return 0;
}
```

➢ **Write a c++ program for local class.**

```
#include<iostream.h>
#include<conio.h>
void demo()
{
	class inner
	{
		int x;
		public:
		void set(int y)
		{
			x=y;
		}
		void disp()
		{
			cout<<"Value of x is: "<<x;
		}
	};

	inner obj;
	obj.set(10);
	obj.disp();
}

void main()
{
	clrscr();
	demo();
	 getch();
}
```

Topic: **What is Constructor?**

- ✓ There is a need to initialize the value of data member before accessing it.
- ✓ Normally member function are used to initialize value of variable.
- ✓ **For Ex: Obj.input(10)**
- ✓ Here, member function input() passes the initialize value as arguments these values are assigned to the private data member of object obj.
- ✓ C++ provides a special member function called the constructor that enables an object to be initialized when it is created.
- ✓ A constructor is different than other member function of the class it has the same name as its class without return type even not void.
- ✓ Similar to other member function the constructor can be defined either within, or outside the body of a class.
- ✓ It can access an data member like all other member function.
- ✓ It must be declared in public part.
- ✓ **Syntax:**

```
class class_name
{
         private:
        data member;
         public:
        class_name()   //constructor
        {
                -------
                -------
        }
        Other member functions;
};
```

- ✓ Here, constructor is defined in a class body by the name of class in public section.
- ✓ Constructor defining outside of class
- ✓ **Ex:**

```
class class_name
{
        private:
        data members;
        public:
        classname();    //constructor declaration
};
classname :: classname()
{
        //body of constructor
}
```

✓ Here, constructor is defined outside the class with the scope of class.
✓ The constructor of a class is the first member function to be executed automatically when object of class is created.
✓ It is executed every time an object is created it can be used to assign initial, value to the data members of the object.

Topic:Give the Characteristics of Constructor?

✓ It has the same name as a class name.
✓ It is executed automatically whenever the class object is created.
✓ It does not have any return type not even void.
✓ It has also default argument as other functions.
✓ It is normally used to initialize the data member of a class.
✓ It is also used to allocate resources such as memory, to dynamic data members of a class.
✓ The address at constructor cannot be referred.
✓ It cannot be virtual.
✓ It cannot be inherited, through derived class call the base class constructor.
✓ It makes use of new and delete operator as implicit call when memory allocation is required.

Topic:What Default Constructor?

- ✓ The default constructor is a special member function with no arguments which initialize the data members.
- ✓ The default constructor accepts no parameters.
- ✓ For ex the default constructor for class data is data() constructor.
- ✓ If no constructor is defined for a class then the compiler supplies the default constructor.

➢ **Write a c++ program for default constructor.**

```
#include<conio.h>
#include<iostream.h>
class demo
{
	int x; public:
	demo()
	{
		cout<<"Demo for constructor....";
		x=10;
	}
	void disp()
	{
		cout<<endl<<"Value of x is: "<<x;
	}
};

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

Topic:What is Parameterized Constructor?

- ✓ The constructor with arguments is called parameterized constructors.
- ✓ It can be invoked same as a function argument by specifying arguments list in brackets.
- ✓ When the constructor is parameterized, must be provide appropriate argument to the constructor.

✓ **Syntax: Class**

```
sample
{
            ---------
            ---------
            Public:
                                sample(int x)
                                {
                                            ------
                                            ------
                                }
--------
       };
```

- ✓ sample s(5);

  or

sample s = sample(5);

- ✓ here, the constructor has a one argument which passed when object of class is created.
- ✓ The passing of initial values to the constructor can be done by two way.

1) Implicit call – sample s(5)
2) Explicit call – sample s = sample(5)

- ➢ **Write a c++ program for parameterized constructor.**

```
#include<iostream.h>
#include<conio.h>
class ABC
{
      private:
          int length,breadth,x;
      public:
          ABC (int a,int b)
          {
                  length = a;

                  breadth = b;
          }
          void area()
          {
                  x = length * breadth;

          }
          void display()
          {
                  cout << "Area = " << x << endl;
          }
};
int main()
{
          clrscr();
          int len,brth;
          cout<<endl<<"Enter length of rect: ";
          cin>>len;
          cout<<endl<<"Enter breadth of rect: ";
          cin>>brth;
          ABC obj(len,brth);
          obj.area();
          obj.display(); getch();
          return 0;
}
```

Topic: Explain multiple constructor in class.

✓ When the class has multiple constructors, is called **constructor overloading.**
✓ All the constructors have the same name as the class in which they belongs to. Same as function overloading.
✓ Each constructor should be differ from their number of arguments and types of arguments.

Topic: Explain constructor with default argument.

✓ It is possible to define constructors with arguments having default value same as any other function in c++.
✓ If any arguments are passed during the creation of an object, the compiler selects the suitable constructor with default argument.
✓ For ex: sample (int i,int j=0);
✓ Here in above constructor the default value of argument j is 0 when create an object.
✓ Sample s(10)
✓ It assigns the value 10 to i and 0 to y by default.
✓ When call the constructor with new value of j it will replaced by new value. Such as
✓ Sample s(10,20)
✓ Now the value of j is consider as 20.

➢ **Write a c++ program for constructor with default argument.**

```
#include<iostream.h>
#include<conio.h>
class demo
{
        int i,j;
        public:
        demo(int a,int b=10)
        {
          i=a;
          j=b;
        }

        void disp()
        {
                cout<<endl<<"Value of i is: "<<i;
                cout<<endl<<"Value of j is: "<<j;
        }
};
void main()
{
        int a;
        clrscr();
        cout<<endl<<"Enter value: ";
        cin>>a;
        demo d(a);
        d.disp();
        getch();
}
```

Topic: Explain copy constructor.

- ✓ A constructor can have argument of any data type.
- ✓ When constructor has object of its own class is called copy constructor.
- ✓ The object of own class must be passed as a reference parameter.
- ✓ For ex: Class

```
test
{
        ------
        ------
public : test(test &obj)
{
        -------
        -------
}
        ------
};
```

✓ Such constructor having a reference to an instance of its own class as an argument is known as copy constructor.

✓ A compiler copies all the members of the user define source object to the destination object in the assignment statement.

✓ For ex:

✓ test t1(5),t2(10);
t1=t2;

✓ here the copy constructor will not invoke. It just assigns the value of t1 to t2, member by member.

✓ This is the task of the overloaded assignment operator (=).

✓ Because both objects are predefined object.

✓ A copy constructor can be called like this way:

```
test t1(5);
test t2(t1);
      or
test t2=t1;
```

✓ Here, in both cases it will invoke copy constructor.

✓ The initialization of one object to another object is performed during object definition.

✓ The data member of t1 is copied to t2 member by member. It is the default action performed by copy constructor.

- **Write a c++ program for constructor with default argument.**

```
#include<iostream.h>
#include<conio.h>
class demo
{
	int i,j;
	public:
	demo(int a,int b=10)
	{
		i=a;
		j=b;
	}
	void disp()
	{
		cout<<endl<<"Value of i is: "<<i;
		cout<<endl<<"Value of j is: "<<j;
	}
};
void main()
{
	int a;
	clrscr();
	cout<<endl<<"Enter value: ";
	cin>>a;
	demo d(a);
	d.disp();
	 getch();
}
```

Topic: **Explain Dynamic Initialization of Object.**

- ✓ A class object can also be initializing at run time. It is known as dynamic initialization of object.
- ✓ One advantage of dynamic initialization is that with the multiple constructors various initialization techniques can be provided.

- ➢ **Write a c++ program for dynamic initialization of object.**

```
#include<iostream.h>
#include<conio.h>
class test
{
        int x; public: test(int a)
        {
                x=a;
        }
        int add()
        {
                return (x*x);
        }
        void disp()
        {
                cout<<endl<<"Value of x is: "<<x;
        }
};

void main()
{
        clrscr();
        int a;
        cout<<endl<<"Enter Value of a:";
        cin>>a;
        test obj1(a);
        obj1.disp();
        test obj2(obj1.add());
        cout<<endl<<"New Object after Square:";
        obj2.disp();
        getch();
}
```

Topic: Explain Dynamic Constructor.

- ✓ A Constructor normally used for the management of memory allocation during runtime.
- ✓ It provides a way to allocate the right amount of memory during execution for each object when the object's data member size is not same.
- ✓ Allocation of memory to objects at the time of their construction is known as dynamic constructor.
- ✓ The constructor which performs such memory allocation is called dynamic constructor.
- ✓ The dynamic memory allocation operator new is used to allocate memory dynamically.

➢ **Write a c++ program for dynamic constructor.**

```
#include<iostream.h>
#include<conio.h>
class sample
{
        int *no,size; public: sample(int s)
        {
                size=s;
                no=new int[size];
        }
        void input();
        void disp();
};

void sample::input()
{
        for(int i=0;i<size;i++)
        {
                cout<<endl<<"Enter Value for "<<i<<": ";
                cin>>no[i];
        }
}

void sample::disp()
{
        for(int i=0;i<size;i++)
        {
                cout<<endl<<no[i];
        }
}
```

```
void main()
{
        clrscr();
        int s;
        cout<<endl<<"Enter Size of Array: ";
        cin>>s;
        sample s1(s);
        s1.input();
        s1.disp();
        getch();
}
```

Topic: What is Destructor?

- ✓ The constructor is called to initialize data member and allocate memory for object at the time of creation.
- ✓ When an object is no longer needed it can be destroyed.
- ✓ Class can have special member function is called destructor.
- ✓ As similar to constructor it automatically gets call when object is destroyed.
- ✓ The destructor has the same name as the class but it specified with ~ complements.
- ✓ Syntax:

```
class clasname
{
         private:
        Member variable;
        public:

                -------

                -------

        ~classname()

        {

        }

};
```

- ✓ Here, shown in above syntax destructor is defined same name with classname.
- ✓ **Rules for defining destructor:**
- ✓ The destructor function has same name as class but prefixed with tiled(~). It makes difference of a constructor and destructor.
- ✓ It has no argument and no return type.
- ✓ Destructor is invoked automatically whenever an object goes out of scope.
- ✓ It is also declare in public section of class .
- ✓ Class can not have more than one destructor.
- ✓ **Difference b'ween constructor and destructor.**
- ✓ Arguments cannot be passed to destructor as constructor.
- ✓ Only one destructor can be declared for a given class while more than one constructor can work in same class.
- ✓ Destructor can not be overloaded as constructor.
- ✓ Destructor can be virtual, while constructor can not be virtual.

➢ **Write a c++ program for destructor.**

```
#include<iostream.h>
#include<conio.h>
class demo
{
	public:
	demo()
	{
		cout<<endl<<"Constructor is called...";
	}
	~demo()
	{
		cout<<endl<<"Destructor is called....";
	}
};
int main()
{
	clrscr();
	demo d;
	getch();
}
```

Topic: What is MIL?

- ✓ MIL stands for Multiple Initialize list.
- ✓ Initialize List is used in initializing the data members of a class.
- ✓ The list of members to be initialized is indicated with constructor as a comma-separated list followed by a colon.
- ✓ Syntax:

```
class Point
{
            private:
            int x;
            int y;

                  public:
                  Point(int i, int j):x(i), y(j) {}
};
```