UNIT-2

Introduction to Classes and Objects: Defining Classes & Objects, Access specifiers, Scope resolution Operator, Static Member variables, Static Member Functions, Array of Objects. Inline Functions, Overloading Member Functions, Objects as Function Arguments, Friend Functions, Friend Class, Local Class, Empty Class, Nested Classes, Return by Reference.

1. Defining Classes & Objects:

Class:

Object Oriented Programming encapsulates data (attributes) and methods (behavior) into package is called class.

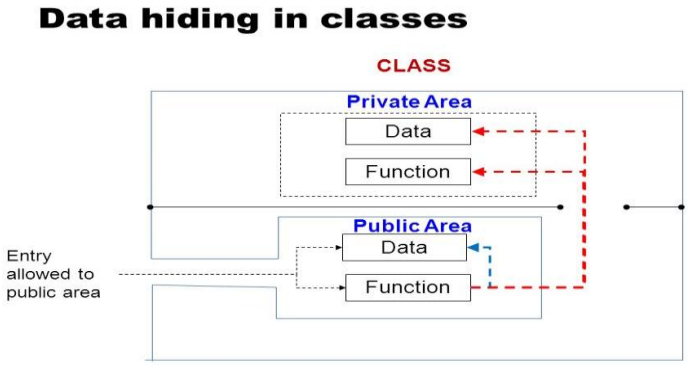
The class combines data and methods for manipulating that data into one package. An object is said to be an instance of a class. A class is a way to bind the data and its associated methods together. It allows the data to be hidden from external use. When defining a class, we are creating a new *abstract data type* that can be treated like any other built-in data type.

Generally, a class specification has two parts:

1. Class declaration
2. method definitions

The **class declaration** describes the type and scope of its members. The method definitions describe how the class methods are implemented. The general form of a class declaration is:

```
class class_name
{
        private:
                variable declarations; fu
                declarations;
        public:
                variable declarations;
                function declarations;
};
```



- ✓ The class declaration is similar to structure declaration in C. The keyword **class** is used to declare a class. The body of a class is enclosed within braces and terminated by a semicolon.

- ✓ The class body contains the declaration of variables and methods. These methods and variables are collectively called members. They are usually grouped under two sections i,e. private and public to denote which members are private and which are public. These keywords are known as **visibility labels**.

- ✓ The members that have been declared as private can be accessed only from within the class. On the other hand, public members can be accessed from outside the class also.

- ✓ The data hiding is the key feature of OOP. By default, the members are **private.** The variables declared inside the class are known as data members and the functions are known as *member functions (or) methods.*

- ✓ Only the member functions can have access to the private members and methods. However, the public members can be accessed from outside the class.

- ✓ The binding of data and methods together into a single class type variable is referred to as *encapsulation.*

A Simple Class Example: **A typical class declaration would look like:**

```
class item

{
        int no;                 // variable declaration int no;
                                   // private by default
    public:
        void getdata(int a, float b); // method declaration
         void putdata(void);          // using prototype
};
```

Objects :

Objects are the basic run-time entities in an object oriented system.Once a class has been declared, we can create variables of that type by using the class name. General form of object declaration is

class_name objectname1,….;

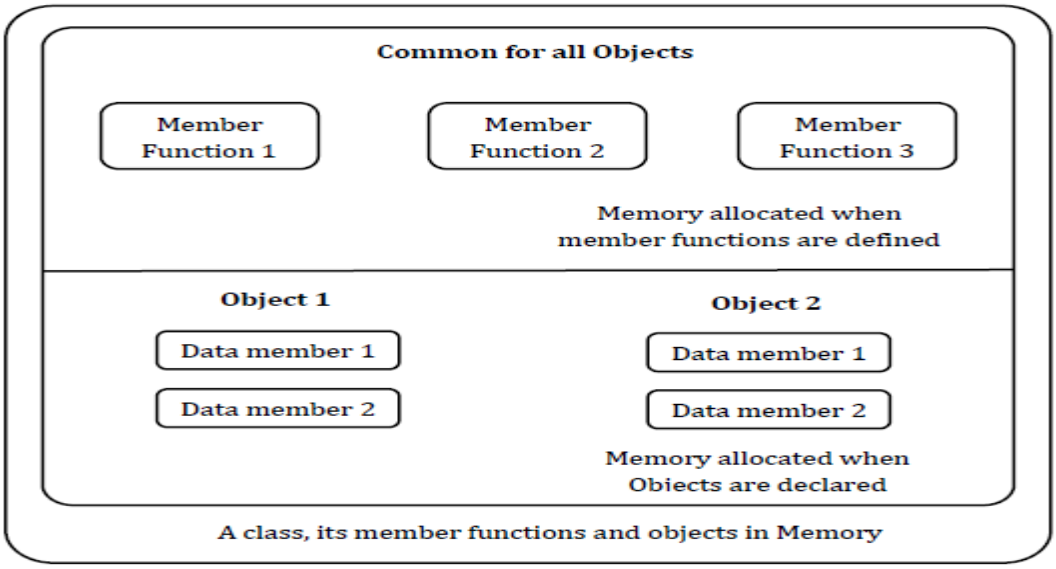
For example,

item x; // memory for x is created Creates a variable x of type *item*.

Memory Allocation for Objects:

• Each object has its own separate data items and memories are allocated when the objects are created.

• Member function are created and put in the memory only once when class are defined.



In C++, the class variables are known as *objects.* Therefore, x is called an object of type item. We may also declare more than one object in one statement. Example:

item x, y, *z;

The declaration of an object is similar to that of a variable of any basic type. The necessary

memory space is allocated to an object at this stage.

Note that class specification provides only a *template* and does not create any memory space for the objects.

How can you access the class members?

The object can access the public class members of a class by using dot operator or arrow operator. The syntax is as follows;

Objectname operator membername;

Example:

x.show();
z->show(); \\z is a pointer to class item

Member functions:

Member functions are defined in two places. They are

1. Inside the class
2. Outside the class

Inside the class

Member functions can be defined immediately after the declaration of member variables inside the class. These functions by default act as **inline functions**.

Example:

```
class student
{
        private:
                int rno;
                char *sname;
        public:
        void print()
        {
                cout<<"rno="<<rno;
                cout<<"name="<<sname;
        }
        void read(int a, char *s)
        {
                rno = a;
                strcpy(sname, s);
        }
};
void main()
{
        student s;
         s.read(10, "Rama");
        s.print();
}
```

```
Output:
 rno= 10
name = Rama
```

Outside the class

- ✓ Member functions that are declared inside a class have to be defined separately outside the class. Their definitions are very much like the normal functions.
- ✓ The main difference is the member function incorporates a membership identity label in the header. This label tells the compiler which class the function belongs to.

General form

```
return-type classname :: function-name (list of arguments)
{
        // function body
}
```

Example:

```
class student
{
        private:
                int rno;
                char *sname;
        public:
                void read(int , char*);
                 void print();
};
void student :: read (int a, char *s)
{
        no = a;
        strcpy(sname, s);
}
void student :: print()
{
        cout<<"rno="<<rno;
        cout<<"name="<<sname;
}
void main()
{
        student s;
        s.read(10, "Rama");
        s.print();
}
```

Output: rno=10

name = Rama

What are the characteristics of member functions?

- ✓ The member functions are accessed only by using the object of the same class.
- ✓ The same function can be used in any number of classes.
- ✓ The private data or private functions can be accessed only by public member functions.

2. Access specifiers:

The access specifier specifies the accessibility of the data members declared inside the class. They are:

1. public
2. private
3. protected

public keyword can be used to allow object to access the member variables or member functions of class directly. The keyword **public** followed by colon (:) means to indicate the data member and member function that visible outside the class.
Consider the following example:

class Test

```
{
    public:
        int x, y;        // variables declaration
        void show()
        {
            cout<<x<<y;
        }
};
int main()
{
    Test t;              //Object creation
    t.x = 10;
    t.y = 20;
    t.show();
}
```

Now, it display 10 and 20

Private keyword is used to prevent direct access to member variables or member functions by the object. It is the **default access**. The keyword **private** followed by colon (:) is used to make data member and member function visible with in the class only.

Consider the following example:

```
class Test
{
    private:
        int x, y;        // variables declaration
};
int main()
{
    Test t;              //Object creation
    t.x = 10;
    t.y = 20;
}
```

Now it raises two common compile time errors:

„Test::x" is not accessible
„Test::y" is not accessible

Protected access is the mechanism same as private. It is frequently used in inheritance. Private members are not inherited at any case whereas protected members are inherited. The keyword **protected** followed by colon (:) is used to make data member and member function visible with in the class and its child classes.

```
class Test
{
    protected:
        int x, y;        // variables declaration
};
 int main()
{
    Test t;  //Object creation
    t.x = 10;
    t.y = 20;
}
```

Now it raises two common compile time errors:

„Test::x“ is not accessible

„Test::y“ is not accessible

- Private – No Direct Access
- Protected – No Direct Access
- Public – Direct Access

Access-control specifiers	Accessible to	
	Own class members	Objects of a class
Private	Yes	No
Protected	Yes	No
Public	Yes	Yes

3. Scope resolution Operator:

The scope resolution operator is used to reference the global variable, define member function that is out of class scope, refer to static members of a class. 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.
For example, when the global and local variable or function has the same name in a program, and when we call the variable, by default it only accesses the inner or local variable without calling the global variable. In this way, it hides the global variable or function. To overcome this situation, we use the scope resolution operator to fetch a program's hidden variable or function.
Ex:

```
#include<iostream.h>
int x=30;//global variable
int main()
{
    int x=20;//local variable
    cout<<“Local variable:”<<x<<endl;
    cout<<“Global variable:”<<::x;
    return 0;
}
```

Uses of the scope resolution Operator

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

4. Static Member variable:

- It is initialized to zero when first object of class is created. No other initialization is required.
- Only one copy is created for entire class and is shared by all the objects of that class.
- It is visible within the class but its life time is the entire program.
- It is associated with the class and not with any object.

Type and scope of each static member variable must be defined outside the class definition. They are stored separately rather than as a part of the object.

```
#include<iostream.h>
#include<conio.h>
class item
{
        static int c;
        int k;
    public:
        void getdata(int a)
        {
            k=a;
            c++;
        }
        void getcount()
        {
            cout<<"Count: "<<c<<endl;
        }
};
int item::count;
// This is mandatory for Static data members of a class, because we know that static members are not stored with class data members.
int main()
{
    item A,B,C;
    clrscr();
    cout<<"Before Reading Data" <<endl;
    A.getcount();
    B.getcount();
    C.getcount();
    A.getdata(10);
    B.getdata(20);
    C.getdata(30);
    A.getcount();
    B.getcount();
    C.getcount();
    return 0;
}
```

Output:
Before Reading Data
Count:0
Count:0
Count:0
Count:3
Count:3
Count:3

5. Static Member Functions:

It has access to only other static members(function or variables) declared in the same class. The non-static members are not available to these functions. The static member function declared in public section can be invoked using its class name without using its object. To declare static function just put the keyword “static” in front of the function definition.

```
static returnType functionName(arguments)
{
    function body;
}
```

It can be called using the class name as follows:

```
className :: functionName(arguments);
```

Ex:

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

```
class item
{
        static int count;
        int code;
    public:
        void setcode()
        {
            code=++count;
        }
        void showcode()
        {
            cout<<"Object Number: "<<code<<endl;
        }
        static void showcount()
        {
            cout<<"Count: "<<count<<endl;
        }
};
int item::count;
void main()
{
    item t1,t2;
    clrscr();
    t1.setcode();
    t2.setcode();
    item::showcount();
    item t3;
    t3.setcode();
    item::showcount();
    t1.showcode();
    t2.showcode();
    t3.showcode();
    getch();
}
```

Output:

Count:2

Count:2

Count:3

Object Number:1

Object Number:2

Object Number:3

Remember the following Won't Work because code is not static.

```
static void showcount()
{
cout<<code; //code variable is not static
}
```

6. Array of Objects:

Arrays are collection of similar data types. Arrays can be of any type including user-defined data type created using struct, class and typedef declarations. Thus, an array of a class type is also known as an array of objects. An array of objects is declared in the same way as an array of any built-in data type. The array elements are stored in continuous memory locations.

```
#include <iostream.h>

class player
{
    char name[10];
    int age;
public:
void input()
```

```
{
cout<<"Enter player name:";
cin>>name;
cout<<"Age:";
cin>>age;
}
void display()
{
cout<<"\n Player name:"<<name;
cout<<"\nAge:"<<age;
}
};
int main()
{
    player cricket[3];
    int i;
    for(i=0;i<3;i++)
    cricket[i].input();
    for(i=0; i < 3; i++)
    cricket[i].display();
    return 0;
}
```

7. Inline Functions:

If make a function as inline, then the compiler replaces the function calling location with the definition of the inline function at compile time.The inline mechanism reduces overhead relating to access the member function.It provides better efficiency and allow quick execution of function.As inline member function is similar to macro.default, all member functions defined inside the class are inline functions.

When the inline function is used:

1)Inline functions should be used rarely. It should be applied only at appropriate circumstances.

2) Inline function can be used when the number function contains few statements.

3)If the function takes more time to execute, then it must be declared as inline.

Some of the situations where By inline expansion may not work are:

1)For function returning value,loops,switch,goto exists.

2)Function contains static variables.

3)If inline functions are recurssive.

Syntax for an inline function:

inline return_type class_name::function_name(parameters)
{
// function code?
}
Ex:

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

```
class account
{
    int acno;
    char acname[20];
    float acbal;
    public :
    void getAccount();//member function declartion
    void putAccount();//member function declartion
};
inline void account :: getAccount() {
    cout<<"Enter Account Number :";
    cin>>acno;
    cout<<"Enter Account Name :";
    cin>>acname;
    cout<<"Enter Account Balance :";
    cin>>acbal;
}
void account ::putAccount() {
    cout<<"Account Number ="<<acno<<endl;
    cout<<"Account Name ="<<acname<<endl;
    cout<<"Account Balance ="<<acbal<<endl;
}
int main()
{
    account ac;
    ac.getAccount() ;
    cout<<"Account Details :"<<endl;
    ac.putAccount() ;
}
```

8. Overloading Member Function:

Member functions can be overloaded like any other normal functions. Overloading means one function is defined with multiple definitions with same functions name in the same scope.The secret to overloading is that each redefinition of the function must use either

- different types of parameters

Ex: sum(int, int, int); sum(float, float, float);

- different number of parameters.

Ex: sum(int, int, int); sum(int, int);

- order of the arguments are different

Ex:sum(int,float,int);sum(float,int,int);

- You cannot overload function declarations that differ only by return type.

```
#include<iostream>
using namespace std;
class overloadDemo
{
        public:
                int add(int x,int y)
                {
                        return x+y;
                }
```

```
        double add(double x,double y)
        {
                return x+y;
        }
        char add(char x,char y)
        {
                return x+y;
        }
};
int main()
{
        overloadDemo l;
        cout<<" The addition of integers is :"<<l.add(3,4)<<endl;
        cout<<" The addition of floats is :"<<l.add(3.7,4.9)<<endl;
        cout<<" The addition of characters is :"<<l.add('a','b')<<endl;
}
```

9. Objects as Function Arguments:

Similar to variables, object can be passed to functions. The following are the three methods to pass arguments to a function.

1)Pass-by-value: A copy of object(actual object) is sent to function and assign to the object of called function(formal object).Both actual and formal copies of objects are stored different memory locations. Hence, changes made in formal object are not reflected to actual object.

```
#include<iostream.h>
class passbval
{
    public:
    int x;
    void add(clbval c)
    {
    c.x=x+c.x;
    }
    void show()
    {
    cout<<x;
    }
};
int main()
{
    passbval c1;
    c1.x=50;
    cout<<"\nBefore c1:";
    c1.show();
    c1.add(c1);
    cout<<"\nAfter C1:";
    c1.show();
    return 0;
}
```

2)Pass-by-reference: Address of object is implicitly sent to function.

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

```
class passbref
{
    public:
    int x;
    void add(clbval &c)
    {
    c.x=x+c.x;
    }
    void show()
    {
    cout<<x;
    }
};
int main()
{
    passbref c1;
    c1.x=50;
    cout<<"\nBefore c1:";
    c1.show();
    c1.add(c1);
    cout<<"\nAfter C1:";
    c1.show();
    return 0;
}
```

3)Pass-by-address: Address of the object is explicitly sent to function.

```
#include<iostream.h>
class passbadd
{
    public:
    int x;
    void add(clbval *c)
    {
    c->x=x+c->x;
    }
    void show()
    {
    cout<<x;
    }
};
int main()
{
    passbadd c1;
    c1.x=50;
    cout<<"\nBefore c1:";
    c1.show();
    c1.add(&c1);
    cout<<"\nAfter C1:";
    c1.show();
    return 0;
}
```

10. Friend Functions:

- The central idea of encapsulation and data hiding concept is that any non-member function has no access permission to the private data of the class.

- The private members of the class are accessed only from member functions of that class.
- C++ allows a mechanism, in which a non-member function has access permission to the private members of the class. This can be done by declaring a non-member function friend to the class whose private data is to be accessed.

```
Class ABC
{
    ……
    ……
    public:
    ……..
    …….
    friend void xyz(ABC);
};
```

- The function is defined elsewhere in the program like a normal C++ function.
- A function can be declared as a friend in any number of classes.
- A friend function not a member function,but full access rights to the private members of the class.

```
class Box
{
        int width,height;
    public:
        void set()
        {
            cout<<"Enter width and height:";
            cin>>width>>height;
        }
        friend int printArea( Box);
};
int printArea( Box b1)
{
    return b1.width*b1.height;
}
int main()
{
    Box b;
    b.set();
    Cout<"area:"<<printArea(b);
}
```

Characteristics:

1)There is no scope restriction for the friend function; hence,they can be called directly without using object.

2)It can be declared either in public or private sections of a class without effecting its meaning.

3)Use of friend is rare,since it violating the rule of encapsulation and data hiding.

4)unlike member functions,it cannot access the member data directly and has to use an object name and dot membership operator with each member data(e.g.A.x)

5)usually,it has the objects as arguments.

- For example, consider a case where two classes, manager and scientist, have been defined. we would like to use a function incometax() to operate on the objects of both the classes.

- In such situation, c++ allows the common function to be made friendly with both the classes, thereby allowing the function to have access to the private data of these classes. such a function need not be a member of any of these classess.
- To make an outside function friendly to a class,we have to simply declare this function as a friend of the class.

```
#include<iostream>
using namespace std;
class first;
class second
{
    int a;
    public:
    void getval()
    {
    cout<<"\n Enter a number:";
    cin>>a;
    }
    friend void sum(second,first); //friend with object as argument
};
class first
{
    int b;
    public:
    void getval()
    {
    cout<<"\n Enter a number:";
    cin>>b;
    }
    friend void sum(second,first);//friend with object as argument
};
void sum(second x,first y)
{
    cout<<"\n Sum of two numebrs:"<<x.a+y.b;
}
int main()
{
    second x1;
    first y1;
    x1.getval();
    y1.getval();
    sum(x1,y1);
}
```

11. Friend Class:

It is possible to declare one or more functions as friend functions or an entire class can also be declared as friend class.

when all the functions need to access another class in such situation we declare an entire class as friend class.

The friend is not transferable or inheritable from one class to another.

Declaring class A to be a friend of class B does not mean class B a friend of class A.

```
#include<iostream>
using namespace std;
class B;
class A
{
        int x;
public:
        void setval()
        {
        x=10;
        }
friend B;
};
class B
{
        int y;
public :
        void setval()
        {
        y=20;
void show(A obj)
{
cout<<"x value ="<<obj.x<<endl;
cout<<"y value="<<y<<endl;
}
};
int main()
{
        A obj1;
        B obj2;
        obj1.setval() ;
        obj2.setval() ;
        obj2.show(obj1) ;
        return 0;
}
```

12. Local Class:

Classes are declared inside the function then such classes are called as local classes.

The local classes have access permission to global variables as well as static variables.

The global variables need to accesses using scope access operator when the class itself contains member variable with same name as global variable.

The local classes should not have static data members and static member functions. if we declared, the compiler provides an error message.

```
#include<iostream.h>
//using namespace std;
int i;
void func()
{
    class LocalClass
    {
    public:
    int i;
    //static int j;
    void getdata()
    {
cout<<"Enter Local and global values:";
    cin>>i>>::i;
    }
void putdata() {
cout<<"The Local number is:"<<i;
cout<<"Global number is:"<<::i;
    }
};
//int LocalClass::j;
LocalClass obj;
obj.getdata();
obj.putdata();
}
int main() {
cout<<"Demonstration of a local class"<<endl;
func();
return 0;
```

```
}
```

13. Empty Class:

Empty class means, a class that does not contain any data members size of empty class is 1 byte

```
#include <iostream>

using namespace std;

class Person

{

};

int main() {

    Person per;

    cout << "size of per: " << sizeof(per) << endl;

return 0;

}
```

14. Nested Classes:

A class is defined in another class, it is also known as nesting of classes.

In nested class the scope of inner class is restricted by outer class.

```
#include<iostream.h>
class outer{
        int od;
        public:
        class inner{
                private:
                    int id;
                public:
                        void getdata(int n)
                        {
                        id = n;
                        }
                        void putdata()
                        {
                        cout<<"The inner class data: "<<id<<endl;
                        }
                };
                        void getdata(int n)
                        {
                        od = n;
                        }
```

```
                    void putdata()
                    {
                    cout<<"The outer class data: "<<od<<endl;
                    }

};
int main() {
        outer o;
        cout<<"Nested classes in C++"<< endl;
        outer::inner i;
        o.getdata(20);
        o.putdata();
        i.getdata(9);
        i.putdata();
        return 0;
}
```

15. Return by Reference:

In C++, Pointers and References held close relation with in another.

The major difference is that the pointers can be operated on like adding values whereas references are just an alias for another variable.Functions in C++ can return a reference as it is return a pointer.When function returns a reference it means it returns a implicit pointer.

In C++ Programming, not only can you pass values by reference to a function but you can also return a value by reference.Return by reference is very different from Call by reference.

Functions behaves a very important role when variable or pointers are returned as reference.

```
#include <iostream.h>
int& max(int &x,int&y){
        if(x>y)
        return x;
        else
        return y;
}
int main(){
        int a=20,b=10;
        max(a,b) = -1;
        cout <<a<<" "<<b;
return 0;
}
```