**OBJECT ORIENTED PROGRAMMING STRUCTURE(OOPS)**

Class and object:

**Class:**

It is an user defined data type . It is an extension of structure.

or

It is a template or blue print which contains data members and functions.

EX:

class sample

{

int x; // datamember

void get(); ----// CLASS FUNCTION OR MEMBER FUNCTION

};

**object:**

It is an entity which shares or draws the common properties of the class and can have its own behaviour.

Examples:

Fruit------->class

Mango, banana,orange-----> objects

Class animal

{

Eye,mouth

See()

Eat()

};

Tiger, cat,dog------->OBJECTS

Class exam

{

What is c++

What is java

}

Student1,student2,student3----->OBJECTS

class bank

{

void deposit()

void balance()

void withdrawl()

};

customer1,customer2,customer3-------->OBJECT

**ACCESS SPECIFIER:**

It specifies the accessibility of the members of the class.

They are

**Private**

**Public**

**Protected**

**Private:**

The members(data member or member function) declared under private category cannot be accessed outside the class.

**Public:**

The members declared under the public category can be accessed outside the class any where in the program.

**Protected:**

The members declared under the protected category can be accessed only to the child class under inheritance.

#include<iostream.h>

#include<conio.h>

class sample

{

public:

int x;

};

void main()

{

sample s;

clrscr();

s.x=100;

cout<<s.x<<endl;

getch();

}

THE ABOVE CLASS AND OBJECT PROGRAM IS SAME AS THAT OF THE STURCTURE PROGRAM OF "C"

#include<iostream.h>

#include<conio.h>

class sample

{

private:

int x;

};

void main()

{

sample s;

clrscr();

s.x=100;

cout<<s.x<<endl;

getch();

}

The above program does not work because the object cannot access private data member.

Data member of the class be declared under private category and member function or class function be declared under public category.

Object can access only the members declared under public category.

Object is not a member of the class. It is a non member.

**MEMBER FUNCTION OR CLASS FUNCTION**:

The function which is a member of the class is called as class function or member function.

NOTE:

By default the members of the structure are public.

To access the members of the class using an object the dot(.) Operator is used.

Oops concept is a bottom--up approach. Because where the object is created from their the program flow or executionstarts .

#include<iostream.h>

#include<conio.h>

class sample

{

private:

int x;

public:

void get()

{

cin>>x;

}

void put()

{

cout<<x<<endl;

}

};

void main()

{

sample s;

clrscr();

s.get();

s.put();

getch();

}

**Program to accept one employee information**

#include<iostream.h>

#include<conio.h>

class employee

{

private:

int empno; empno

100

char empname[10];

float empsal; empname

ajay

public:

void getdata() empsal

25000

{

cout<<"enter the employee details"<<endl; e1

cin>>empno>>empname>>empsal;

}

void display()

{

cout<<"The details of an employee is"<<endl;

cout<<empno<<" "<<empname<<" "<<empsal;

}

};

void main()

{

employee e1;// e1 IS AN OBJECT OF CLASS employee

clrscr();

e1.getdata();

e1.display();

getch();

}

**output:** enter the employee details

100 ajay 25000

the details of an employee is

100 ajay 25000

**Program to accept two employee information**

#include<iostream.h>

#include<conio.h>

class employee

{

private: empno empno

101

100

int empno;

char empname[10]; empname empname

ravi

raju

float empsal;

public: empsal empsal

32000.50

25000.25

void getdata()

{

cout<<"enter the employee details"<<endl;

cin>>empno>>empname>>empsal;

}

void display()

{

cout<<"The details of an employee is"<<endl;

cout<<empno<<" "<<empname<<" "<<empsal;

}

};

void main()

{

employee e1,e2;// e1 IS AN OBJECT OF CLASS employee

clrscr();

e1.getdata();

e2.getdata();

e1.display();

e2.display();

getch();

}

**output:** enter the employee details

100 raju 25000.25

**output:** enter the employee details

101 ravi 32000.50

The details of an employee is

100 raju 25000.25

The details of an employee is

101 ravi 32000.50

**ARRAY OF OBJECTS**:

Instead of declaring "n" number of objects , declare the objects in the form of an array.

EX: student s[10]-------> ARRAY OF 10 OBJECTS.

employee e[10] // ARRAY OF 10 EMPLOYEE OBJECT

#include<iostream.h>

#include<conio.h>

class employee

{

private:

int empno;

char empname[10];

float empsal;

public:

void getdata()

{

cout<<"enter the employee details"<<endl;

cin>>empno>>empname>>empsal;

}

void display()

{

cout<<"The details of an employee is"<<endl;

cout<<empno<<" "<<empname<<" "<<empsal<<endl;

}

};

void main()

{

employee e[5]; // ARRAY OF 10 EMPLOYEE OBJECT

int i,n;

clrscr();

cout<<"enter the no of employees"<<endl;

cin>>n;

for(i=0;i<n;i++)

e[i].getdata();

for(i=1;i<=n;i++)

e[i].display();

getch();

}

Memory

e[0] e[1] e[2] e[3] e[4]

garbage

values in 3 location

104

Nithish

50000

103

vishwas

35000

102

vijay

30000

101

vinay

25000

empno

empname

NOTE:

BY DEFAULT THE MEMBERS OF THE CLASS ARE PRIVATE.

empsal

SCOPE OF THE FUNCTION DEFINITION OUTSIDE THE CLASS:

output:

enter the no of employees

4

enter the employee details

101 vinay 25000

enter the employee details

102 vijay 30000

enter the employee details

103 vishwas 35000

enter the employee details

104 nithish 50000

the details of an employee is

101 vinay 25000

102 vijay 30000

103 vishwas 35000

104 nithish 50000

Functions can be declared and defined inside the class.

Functions can also be declared inside and defined outside the class.

While defining outside , the function must be preceded with scope resolution operator(::) and classname with data type

This is scope of the function definition outside the class.

Note:

It is used to formatting the informations of the class.

class demo

{

int x;

public :

void get()

{

}

};

EX:

class demo

{

public:

void get(); --------------------> DECLARATION OR PROTOTYPE

};

void demo::get() --------->DEFINITION

{

}

// SCOPE OF THE FUNCTION DEFINITION OUTSIDE THE CLASS

#include<iostream.h>

#include<conio.h>

class student

{

private:

int rollno;

char name[10];

float marks;

public:

void input();

void output();

};

void student::input()

{

cin>>rollno>>name>>marks;

}

void student::output()

{

cout<<rollno<<" "<<name<<" "<<marks;

}

void main()

{

student s;

clrscr();

s.input();

s.output();

getch();

}

// SCOPE OF THE FUNCTION DEFINITION OUTSIDE THE CLASS

#include<iostream.h>

#include<conio.h>

class arithmetic

{

private:

int a,b;

public:

void getdata();

void putdata();

int add();

int sub();

int mul();

int div();

};

void arithmetic::getdata()

{

cout<<"enter the two values"<<endl;

cin>>a>>b;

}

void arithmetic::putdata()

{

cout<<"The addition of two values"<<add()<<endl;

cout<<"The subtraction of two values"<<sub()<<endl;

cout<<"The multiplication of two values"<<mul()<<endl;

cout<<"The division of two values"<<div()<<endl;

}

int arithmetic::add()

{

return a+b;

}

int arithmetic::sub()

{

return a-b;

}

int arithmetic::mul()

{

return a\*b;

}

int arithmetic::div()

{

return a/b;

}

void main()

{

arithmetic obj;

clrscr();

obj.getdata();

obj.putdata();

getch();

}

**Function overloading :**

A function having same name,but different signature( differ in the number of arguments or data type of argument).

C++ supports a special kind of functions called overloaded functions. Two or more function definitions having the same name, declared in the same scope but with different parameters are called overloaded functions.

When an overloaded function is invoked, the c++ compiler chooses the appropriate function by examining the number, data types, and order of the arguments present in that function call. Overloaded function are used to perform similar tasks, but on different data types.

We can overload two or more function by doing two things

* Give the same name to all functions
* provide a parameter list which is unique in either the number of parameters or their data types.

eg:

Int add(int val1, int valu2);

Float add(float fval1, float fval2);

Here the function name add is common for both functions but there signatures are different

**Example1 :**

# include<iostream.h>

int sum(int,int);

float sum(int ,float);

void sum(float,int);

void main()

{

Sum(5.2,6);

Sum(5,3.2); // second function call

Sum(6,2);

}

int sum(int x,int y)

{

Cout<<”first function”<<x+y<<”\n”;

}

float sum(int a, float b)

{

Cout<<”second function\n”<<x\*y<<”\n”;

}

void sum( float a, int b)

{

Cout<<”third function”<<x+y<<”\n”;

}

**output:**

third function 11.2

second function 16.0

third function 8

**Argument type conversions**

Once the viable functions are selected converting the arguments to their corresponding parameters types, is done by the compiler. It yields the following situation.

**An Exact match** : In an exact match situation, the argument in a function call is exactly the same as the data type of its corresponding parameter of the variable function and number of parameter. Hence the name exact match.

**A match with a type conversion**: when there no exact math of functions but an argument of a function call matches exactly the function parameters, then it requires conversions to make its type as that of function parameter

default promotion: 1) char to int ,float to double

2)built in conversion : int to float

float to int

#include<iostream.h>

#include<conio.h>

float sum(float a,int b) //first function

{

float c=a+b;

return(c);

}

float sum(int a,int b) //second function

{

float c=a\*b;

return(c);

}

void main()

{

int a=4;

float b=5.8;

cout<<sum(a,b);

}

**output: 20**

second function will call because no matching type is found i.e (int,float) in the above prg. In second function (int,float) is their, float convert to int in second parameter i.e match type conversion.

**No match** : This situation arises when there is no match in function call arguments and called function parameters

**Friend function**

We know that the private member data can be accessed by the member function only. Under some circumstance non member functions also be able to acess the private member data of some class. This is where the concept of friend function come into the picture.

It is a non member function which is used to access the private data members of the class.

Any function which is preceded with the keyword "friend" is called friend function.

ex: friend void display() ------> FRIEND FUNCTION. ( NON MEMBER FUNCTION)

#include<iostream.h>

#include<conio.h>

class sample

{

private:

int x;

public:

void getdata();

friend void display(class sample); // NON MEMBER FUNCTION, FRIEND

};

void sample::getdata()

{

cout<<"enter the value for x"<<endl;

cin>>x;

}

void display(sample a)

{

cout<<a.x<<endl;

}

void main()

{

sample s;

clrscr();

s.getdata();

display(s);

getch();

}

**NOTE:**

Though the friend function is declared inside the class, it is not a member of

Friend function should not be called by using object because it is not a member of the class.

It should be called directly by specifying the name of the function.

NOTE:

Friend function can be declared and defined inside the class irrespective of any category(private , public etc).

**Adding private data members of two classes using common friend function**

#include<iostream.h>

#include<conio.h>

class beta;

class alpha

{

private:

int x;

public:

void accept();

friend int sum(class alpha, class beta);

};

class beta

{

private:

int y;

public:

void input();

friend int sum(class alpha,class beta);

};

void first::accept()

{

cout<<"enter the value for x"<<endl;

cin>>x;

}

void second::input()

{

cout<<"enter the value for y"<<endl;

cin>>y;

}

int sum(alpha a, beta b)

{

return(a.x+b.y);

}

void main()

{

alpha a;

beta b;

clrscr();

a.accept();

b.input();

int m= sum(a,b);

cout<<m<<endl;

getch();

}

**OUTPUT**

enter the value of x

2

enter the value of y: 3 sum=5

**FRIEND CLASS**

**Making one class friend to another class :**

When the class second is made friend to class first, then the private data member of class first can be accessed inside the member function of class second.

But vice versa is not possible.

#include<iostream.h>

#include<conio.h>

class first

{

private:

int x;

public:

void input()

{

cout<<"enter the value for x"<<endl;

cin>>x;

}

friend class second; // FORWARD DECLARATION

};

class second

{

public:

void output(first f)

{

cout<<f.x<<endl;

}

};

void main()

{

first obj1;

second obj2;

clrscr();

obj1.input();

obj2.output(obj1);

getch();

}

**Constructor**

Every object created would have a copy of member data which requires initialization before its use. C++ allows objects to initialize themselves as and when they are created. This automatic initialization is performed through the use of constructor functions. Destructors are functions that are complimentary constructors.

**Constructor**

It is a special member function which is automatically called when an object is created

Characteristics of constructor

* A constructor has the same name as that of the class
* A constructor is declared in the public section of a class definition
* A constructor is invoked automatically as soon as the class object is created.
* A constructor does not return any value, so return type is not associated with its definition (even void is not used).
* The programmer has no direct control over constructor functions
* The constructor is not inherited, but a derived class can call the base class constructor.
* A constructor cannot be declared as virtual
* Constructor can be overloaded.

**Default constructor**

Default constructor is a type of function which is used to initialize the values

Example

Class box

{

Private:

int length,width,height;

Public:

box()

{

cout<<”enter length,width and height\n”;

cin>>length>>width>>height;

}

void display()

{

cout<<”length=”<<length<<”width=”<<width<<”height=” <<height<<endl;

}

};

int main()

{

box a;

a.display();

}

* In the above example box is a class with data members length, width and height.
* Along with that it contains member function box which is used to initialize the values of length, width and height
* Here the function name is as same as the class name box.
* The function box does not contain any return value and it is declared under public access specifier.
* In this example function box does not contain any arguments so called default constructor.

A class object of type box is created by the declaration

box a;

And the constructor box() is invoked automatically when a is created and initializes the three data members .

**Parameterized constructor**

Parameterized constructor is also a type of function which is used to initialize the data members with user given values

We can define and declare parameterized constructors. This will be having one or more parameters that receive the argument values. We can change the constructor box() to:

Example

Class box

{

Private:

int length,width,height;

Public:

box(int a,int b,int c)

{

length=a; width=b; height=c;

}

void display()

{

cout<<”length=”<<length<<”width=”<<width<<”height=” <<height<<endl;

}

};

int main()

{

box b1(10,20,30);

b1.display();

}

output: length=10, width=20,height=30

Then if we want to pass valued to constructors through objects can be done by the following:

Box b1(10,20,30);

This statement creates an object name b1 and passes three values 10,20, and 30 to it.

**Copy constructor**

Copy constructor is a parameterized constructor using which one object can be copied to another object. Copy constructors are used in the following situations.

 To initialize an object with the values of already existing objects.

 When objects must be returned as function values.

 To state objects as pass-by- value parameters of a function.

Copy constructor can accept a single argument of reference to same class type. The argument must be passed as a constant reference type.

Syntax: classname :: classname(classname& object)

If there is more than one argument present in the copy constructor, it must contain default arguments.

#include<iostream.h>

class sample

{

int i,j;

public:

sample(int a,int b)

{

i=a;

j=b;

}

sample(sample &s) // CALLING COPY CONSTRUCTOR

{

i=s.i+2;

j=s.j+2;

cout<<endl<<"copy constructor working"<<endl;

}

void put()

{

cout<<i<<" "<<j<<endl;

}

};

void main()

{

sample s1(10,20);

sample s2(s1); // CALLING COPY CONSTRUCTOR

sample s3=s1; // CALLING COPY CONSTRUCTOR

s1.put();

s2.put();

s3.put();

getch();

}

output:

10 20

12 22

10 20

**Constructor overloading**

Besides performing the role of member data initialization, constructors are no different from other functions. This includes overloading also. A class may contain multiple constructors. The concept of multiple constructors in a class provides more flexibility .When multiple constructor used then a proper constructor will be invoked.

class sample

{

int x,y;

public: sample()

{

cout<<”enter 2 values\n”;

cin>>x>>y;

}

sample(int a,int b)

{

x=a; y=b;

}

sample( sample& s)

{

x=s.x;

y=s.y;

}

void display()

{

cout<<”x=”<<x<<”y=”<<y<<endl;

}

int main()

{

sample s1,s2(5,3);

sample s3=s2;

s1.display();

s2.display();

s3.display();

}

output: enter 2 values

10 12

x=10 y=12, x=5 y=3 , x=10 y=12.

**Destructor**

These are the functions that are complimentary to constructors. These are used to de-initialize objects when they are destroyed. A destructor is called when an object of the class goes out of scope, or when the memory space used by it is de allocated with the help of delete operator.

A destructor is a special member function which is also having the same name as that of its class but prefixed with the tilde(~) symbol.

~ name\_of\_the\_class()

{

}

Characteristics of a destructor

* A destructor is invoked automatically by the compiler upon exit from the program and cleans up the memory that is no longer needed.
* A destructor does not return nay value
* A destructor cannot be declared as static, const and volatile.
* A destructor does not accept arguments and therefore it cannot be overloaded.
* A destructor must be declared in public section of a class

**Operator Overloading**

Each C++ operator has a predefined meaning. Most of them are given additional meaning through the concept called operator overloading. The main idea behind operator overloading is to use c++ operators with the class objects. As we apply the c++ operators to the built-in data types, in the same way we can apply those operators to the class objects.

The ability to create new data types, on which direct operations can be performed is called as extensibility and the ability to associate an existing operator with a member function and use it with the objects of its class, as its operands, is called as Operator Overloading.

Eg:

The + (additional operator) may be used to add two int’s , floats or doubles. Now, the same + can be used to add two class objects, thereby + gets overloaded.

When a + operator is used with two ints has different meaning as compared when it is applied to object manipulation.

Advantages

There are many advantages of using operator overloading in programs. It makes the program more natural to read, write and debug.

Limitation

Care should be taken while using operator overloading because if we do not use it properly, it makes our program more difficult to understand.

Implementing Operator Functions

Operator overloading is carried out with the help of a special kind of function. This function is just like any other C++ function definition except that the name of the function consists of the keyword operator followed by the one of the C++’s built in operator. Its general format is

Return\_type operator opr (argument\_list)

{

Function body

}

Return\_type -> is the data type of the return value

Class\_name -> Name of the class

Operator -> is a keyword

Opr -> is a c++’s built in operator.(unary or binary operator)

Argument\_list -> is a list of arguments to be passed

Eg:

int operator + (int s)

{

body of the function ;

}

Here, + operator is overloaded to add two objects of the class

**Overloadable operators**

Assignment operators (=)

Arithmetic operators ( +, -, \* ,/ , %, +=, -=, \*=, /=, %=)

Relational operators ( <, <=, >, >=, ==, !=)

Logical Operators (!, &&, ||)

Bitwise operators ( &, |, ~, ^, <<, >>, <<=, >>=, &=, |=, ^=)

Unary operators (++ , --)

**Operators that cannot be overloaded**

Scope resolution operator : :

Class member access operator .

Class member access operator .\*

Conditional operator ?:

**Function contains only one argument for unary operator.**

**two arguments for binary operator.**

**While a member function has no argument for unary operator and only one for binary operators.**

Types of operator:

Unary operator

Binary operator

Unary operator:

An operator which operates on single operand or variable.

Ex: ++ (increment) and --( decrement ) operators

Ex: ++a , --b

Binary operator:

An operator which operates on two operands or variables.

Ex: arithmetic and relational operators

a+b, a/b

#include<iostream.h>

#include<conio.h>

class assign

{

private:

int x;

public:

void getdata()

{

cin>>x;

}

void display()

{

cout<<x<<endl;

}

};

void main()

{

assign obj1,obj2;

clrscr();

obj1.getdata();

obj2.getdata();

obj1=obj2;

obj1.display();

obj2.display();

getch();

}

// above program uses conventional technique.

The above program uses conventional technique.

That is assigning one object to another.

//USING OPERATOR OVERLOADING ( ARITHMETIC OPERATOR)

#include<iostream.h>

#include<conio.h>

class arithmetic

{

private:

int x;

public:

void getdata()

{

cin>>x;

}

int operator+(arithmetic obj)

{

int temp = x + obj.x;

return temp;

}

};

void main()

{

arithmetic obj1,obj2;

clrscr();

obj1.getdata();

obj2.getdata();

int m = obj1 + obj2; // int m = obj1.operator+(obj2);

cout<<m<<endl;

getch();

}

operator overloading for returning an object

#include<iostream.h>

#include<conio.h>

class arithmetic

{

private:

int x,y;

public:

void getdata()

{

cout<<”Enter 2 numbers\n”;

cin>>x>>y;

}

arithmetic operator+(arithmetic p)

{

p.x= x + p.x;

p.y= y+p.y;

return(p);

}

void display()

{

cout<<”x=”<<x<<”y=”<<y<<endl;

}

};

void main()

{

arithmetic obj1,obj2,obj3;

clrscr();

obj1.getdata();

obj2.getdata();

obj3 = obj1 + obj2; // int m = obj1.operator+(obj2);

obj1.display();

obj2.display();

obj3.display();

getch();

}

**output**

Enter 2 numbers : 5 6

enter 2 numbers : 10 11

x=5 y=6

x=10 y=11

x=15 y=17