Banking Application

#include <iostream>

using namespace std;

// class

class Bank {

private:

int acno;

char name[30];

long balance;

public:

void OpenAccount()

{

cout << "Enter Account Number: ";

cin >> acno;

cout << "Enter Name: ";

cin >> name;

cout << "Enter Balance: ";

cin >> balance;

}

void ShowAccount()

{

cout << "Account Number: " << acno << endl;

cout << "Name: " << name << endl;

cout << "Balance: " << balance << endl;

}

void Deposit()

{

long amt;

cout << "Enter Amount U want to deposit? ";

cin >> amt;

balance = balance + amt;

}

void Withdrawal()

{

long amt;

cout << "Enter Amount U want to withdraw? ";

cin >> amt;

if (amt <= balance)

balance = balance - amt;

else

cout << "Less Balance..." << endl;

}

};

int Bank::Search(int a)

{

if (acno == a) {

ShowAccount();

return (1);

}

return (0);

}

// main code

int main()

{

Bank C[3];

// Bank obj1,obj2,obj3;

// obj1.OpenAccount();

// obj2.OpenAccount();

// obj3.OpenAccount();

int found = 0, a, ch, i;

for (i = 0; i <= 2; i++) {

C[i].OpenAccount();

}

do {

// display options

cout << "\n\n1:Display All\n2:By Account No\n3:Deposit\n4:Withdraw\n5:Exit" << endl;

// user input

cout << "Please input your choice: ";

cin >> ch;

switch (ch) {

case 1: // displating account info

for (i = 0; i <= 2; i++) {

C[i].ShowAccount();

}

break;

case 2: // searching the record

cout << "Account Number? ";

cin >> a;

for (i = 0; i <= 2; i++) {

found = C[i].Search(a);

if (found)

break;

}

if (!found)

cout << "Record Not Found" << endl;

break;

case 3: // deposit operation

cout << "Account Number To Deposit Amount? ";

cin >> a;

for (i = 0; i <= 2; i++) {

found = C[i].Search(a);

if (found) {

C[i].Deposit();

break;

}

}

if (!found)

cout << "Record Not Found" << endl;

break;

case 4: // withdraw operation

cout << "Account Number To Withdraw Amount? ";

cin >> a;

for (i = 0; i <= 2; i++) {

found = C[i].Search(a);

if (found) {

C[i].Withdrawal();

break;

}

}

if (!found)

cout << "Record Not Found" << endl;

break;

case 5: // exit

cout << "Have a nice day" << endl;

break;

default:

cout << "Wrong Option" << endl;

}

} while (ch != 5);

return 0;

}

Book Application

#include<iostream>

using namespace std;

class book{

private:

string author;

string title;

int price;

string publisher;

int stock;

int s\_transaction, f\_transaction;

public:

book(string aut,string tit,int pri,string pub,int sto,int st,int ft);

void buy();

void change\_price();

void stransaction();

void ftransaction();

};

book::book(string aut,string tit,int pri,string pub,int sto,int st,int ft){

author = aut;

title = tit;

price = pri;

publisher = pub;

stock = sto;

s\_transaction = st;

f\_transaction = ft;

}

void book::buy(){

string titl;

cout<<"Enter the title of the book"<<endl;

getline(cin>>ws,titl);

string aut;

cout<<"Enter Author Name : "<<endl;

getline(cin>>ws,aut);

if(titl==title && aut==author){

cout<<"Book Name : "<<title<<endl;

cout<<"Author Name : "<<author<<endl;

cout<<"Price of the book : "<<price<<endl;

cout<<"Current Stock Available : "<<stock<<endl;

int copy;

cout<<"Enter the number of copies required : "<<endl;

cin>>copy;

if(stock>copy){

int pri;

pri = price\*copy;

cout<<"Grand Total = "<<pri<<endl;

stock = stock-copy;

book::stransaction();

}

else{

cout<<"Requested Copies Not In stock"<<endl;

book::ftransaction();

}

}

else{

cout<<"Book is Not Available"<<endl;

}

}

void book::change\_price(){

int pr;

string nam;

cout<<"Enter the book name : "<<endl;

getline(cin>>ws,nam);

if(nam==title){

cout<<"Enter New Price : "<<endl;

cin>>pr;

price = pr;

}else{

cout<<"Book Not available"<<endl;

}

}

void book::stransaction(){

s\_transaction+=1;

cout<<"Successful Transaction = "<<s\_transaction<<endl;

}

void book::ftransaction(){

f\_transaction-=1;

cout<<"Unsuccessul Transaction = "<<f\_transaction<<endl;

}

int main()

{

string aut;

string tit;

int pri;

string pub;

int sto;

int st;

int ft;

int opt;

int i=0;

while(opt!=4){

cout<<"1) Add Book details 2) Buy Book 3) Change Price Of Book 4) Exit"<<endl;

cout<<"Enter your option : "<<endl;

cin>>opt;

if(opt==1){

cout<<"Enter Author Name : "<<endl;

getline(cin>>ws,aut);

cout<<"Enter Book Title : "<<endl;

getline(cin>>ws,tit);

cout<<"Enter Price Of the Book : "<<endl;

cin>>pri;

cout<<"Enter Publisher Name : "<<endl;

getline(cin>>ws,pub);

cout<<"Enter The Stock Available : "<<endl;

cin>>sto;

cout<<"Enter Successful Transaction : "<<endl;

cin>>st;

cout<<"Enter Failure Transaction : "<<endl;

cin>>ft;

}

book obj(aut,tit,pri,pub,sto,st,ft);

if(opt==2){

obj.buy();

}

else if(opt==3){

obj.change\_price();

}

else if(opt==4){

break;

}

i+=1;

}

}

Constructor example

#include<iostream>

using namespace std;

class student

{

int rno;

char name[50];

double fee;

public:

student()

{

cout<<"Enter the RollNo:";

cin>>rno;

cout<<"Enter the Name:";

cin>>name;

cout<<"Enter the Fee:";

cin>>fee;

}

void display()

{

cout<<endl<<rno<<"\t"<<name<<"\t"<<fee;

}

};

int main()

{

student s; //constructor gets called automatically when we create the object of the class

s.display();

return 0;

}

Constructor with array

#include <iostream>

//#define N 5

using namespace std;

class Test {

int x, y;

public:

Test()

{

cout<<"enter the value of x";

cin>>x;

cout<<"enter the value of y";

cin>>y;

}

void print()

{

cout << x << " " << y << endl;

}

};

int main()

{

Test arr[2];

int n = 2;

// for (int i = 0; i < n; i++) {

// arr[i] = Test();

// }

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

arr[i].print();

}

return 0;

}

Friend function:

Example-1

#include <iostream>

using namespace std;

class Box

{

private:

int length;

int breadth;

int height;

public:

void get();

friend void print(Box);

};

void Box :: get()

{

length = 10;

breadth = 20;

height = 30;

}

void print(Box a)

{

cout << "The value of the length is: " << a.length << endl;

cout << "The value of the breadth is: " << a.breadth << endl;

cout << "The value of the height is: " << a.height << endl;

}

int main()

{

Box b;

b.get();

print(b);

return 0;

}

Example-2

#include <iostream>

using namespace std;

class B; // forward declarartion.

class A

{

int x;

public:

void setdata(int i)

{

x=i;

}

friend void min(A,B); // friend function.

};

class B

{

int y;

public:

void setdata(int i)

{

y=i;

}

friend void min(A,B); // friend function

};

void min(A a,B b)

{

int sum = a.x+b.y;

cout<<sum;

// if(a.x<=b.y)

// cout << a.x << std::endl;

// else

//cout << b.y << std::endl;

}

int main()

{

A a;

B b;

a.setdata(10);

b.setdata(20);

min(a,b);

return 0;

}

Operator overloading – unary operator – using member function

#include<iostream>

using namespace std;

class unary

{

private:

int x,y;

public:

void input(int,int);

void output();

unary operator++();

};

void unary :: input(int a,int b)

{

x = a;

y = b;

}

void unary :: output()

{

cout<<x<<"\t"<<y<<endl;

}

unary unary :: operator++()

{

x = ++x;

y = ++y;

}

int main()

{

unary obj;

obj.input(10,20);

obj.output();

++obj;

obj.output();

return 0;

}

Operator loading – unary operator – using friend function

#include<iostream>

using namespace std;

class unary

{

private:

int x,y;

public:

void input(int,int);

void output();

friend unary operator-(unary&);

};

void unary :: input(int a,int b)

{

x = a;

y = b;

}

void unary :: output()

{

cout<<x<<"\t"<<y<<endl;

}

unary operator-(unary &ob)

{

ob.x = -ob.x;

ob.y = -ob.y;

}

int main()

{

unary obj;

obj.input(10,20);

obj.output();

-obj;

obj.output();

return 0;

}

Operator overloading – binary operator – member function

#include<iostream>

using namespace std;

class binary

{

int m1,m2;

public:

void getmarks(int,int);

void putmarks();

binary operator+(binary);

};

void binary :: getmarks(int a,int b)

{

m1 = a;

m2 = b;

}

void binary :: putmarks()

{

cout<<m1<<"\t"<<m2<<endl;

}

binary binary :: operator +(binary obj)

{

binary ob;

ob.m1 = m1+obj.m1;

ob.m2 = m2+obj.m2;

return ob;

}

int main()

{

binary obj1,obj2,obj3;

obj1.getmarks(10,20);

obj2.getmarks(30,40);

obj3 = obj1+obj2;

obj1.putmarks();

obj2.putmarks();

obj3.putmarks();

return 0;

}

Operator overloading – binary operator – friend function

#include<iostream>

using namespace std;

class binary

{

int m1,m2;

public:

void getmarks(int,int);

void putmarks();

friend binary operator+(binary,binary);

};

void binary :: getmarks(int a,int b)

{

m1 = a;

m2 = b;

}

void binary :: putmarks()

{

cout<<m1<<"\t"<<m2<<endl;

}

binary operator +(binary ob1,binary ob2)

{

binary ob;

ob.m1 = ob1.m1+ob2.m1;

ob.m2 = ob1.m2+ob2.m2;

return ob;

}

int main()

{

binary obj1,obj2,obj3;

obj1.getmarks(10,20);

obj2.getmarks(10,20);

obj3 = obj1+obj2;

obj1.putmarks();

obj2.putmarks();

obj3.putmarks();

return 0;

}

Function overloading: addition of one dimensional and two dimensional array

#include<iostream>

using namespace std;

class addition

{

int arr[5];

int mat1[2][2];

int sum;

public:

void add(int []);

void add(int [][2]);

void display();

};

void addition :: add(int num[])

{

sum=0;

for (int i=0;i<5;i++)

sum += num[i];

cout<<sum<<endl;

}

void addition :: add(int m[][2])

{

for(int i=0;i<2;i++)

for(int j=0;j<2;j++)

cout<<m[i][j]<<"\t";

cout<<endl;

}

int main()

{

addition obj;

int num[] = {10,10,10,10,10};

int a1[][2] = {{1,1},{2,2}};

obj.add(num);

obj.add(a1);

return 0;

}

Inheritance – single inheritance

Sample-1

#include<iostream>

using namespace std;

class student

{

private:

int rollno;

int mark1;

public:

int mark2;

void getmarks();

int retmark1();

void putmarks();

};

class result : public student

{

private:

int total;

public:

void sum();

void display();

};

void student :: getmarks()

{

mark1=10;

mark2=20;

}

void student :: putmarks()

{

cout<<"The mark1 and mark2 are:"<<endl;

cout<<mark1<<"\t"<<mark2<<endl;

}

int student :: retmark1()

{

return mark1;

}

void result :: sum()

{

total = mark2+retmark1();

}

void result :: display()

{

cout<<"the total mark is: "<<total;

}

int main()

{

result res;

res.getmarks();

res.putmarks();

res.sum();

res.display();

return 0;

}

Sample-2

#include<iostream>

using namespace std;

class student

{

private:

int mark1;

public:

int mark2;

void getmarks();

int retmark1();

void putmarks();

};

class result : private student

{

private:

int total;

public:

void sum();

void display();

};

void student :: getmarks()

{

mark1=10;

mark2=20;

}

void student :: putmarks()

{

cout<<"The mark1 and mark2 are:"<<endl;

cout<<mark1<<"\t"<<mark2<<endl;

}

int student :: retmark1()

{

return mark1;

}

void result :: sum()

{

getmarks();

total = mark2+retmark1();

}

void result :: display()

{

putmarks();

cout<<"the total mark is: "<<total;

}

int main()

{

result res;

// res.getmarks();

// res.putmarks();

res.sum();

res.display();

return 0;

}

Multilevel inheritance

#include<iostream>

using namespace std;

class student

{

protected:

int rollno;

public:

void getno(int);

void putno();

};

class test : public student

{

protected:

int mark1;

int mark2;

public:

void getmarks(int,int);

void putmarks();

};

class result : public test

{

private:

int total;

public:

void display();

};

void student :: getno(int no)

{

rollno = no;

}

void student :: putno()

{

cout<<rollno<<endl;

}

void test :: getmarks(int m1,int m2)

{

mark1 = m1;

mark2 = m2;

}

void test :: putmarks()

{

cout<<mark1<<"\t"<<mark2<<endl;

}

void result :: display()

{

total = mark1+mark2;

cout<<total;

}

int main()

{

result res;

res.getno(101);

res.putno();

res.getmarks(10,20);

res.putmarks();

res.display();

return 0;

}

Multiple inheritance

#include<iostream>

using namespace std;

class student

{

protected:

int rollno;

public:

void getno(int);

void putno();

};

class test

{

protected:

int mark1;

int mark2;

public:

void getmarks(int,int);

void putmarks();

};

class result : public student, public test

{

private:

int total;

public:

void display();

};

void student :: getno(int no)

{

rollno = no;

}

void student :: putno()

{

cout<<rollno<<endl;

}

void test :: getmarks(int m1,int m2)

{

mark1 = m1;

mark2 = m2;

}

void test :: putmarks()

{

cout<<mark1<<"\t"<<mark2<<endl;

}

void result :: display()

{

total = mark1+mark2;

cout<<total;

}

int main()

{

result res;

res.getno(101);

res.putno();

res.getmarks(10,20);

res.putmarks();

res.display();

return 0;

}

Hierarchical inheritance

#include<iostream>

using namespace std;

class student

{

protected:

int rollno;

int mark1;

int mark2;

int total;

public:

void getno(int);

void getmarks(int,int);

};

class details : public student

{

public:

void putno();

};

class test : public student

{

public:

void putmarks();

void display();

};

void student :: getno(int no)

{

rollno = no;

}

void student :: getmarks(int m1,int m2)

{

mark1 = m1;

mark2 = m2;

}

void details :: putno()

{

cout<<rollno<<endl;

}

void test :: putmarks()

{

cout<<mark1<<"\t"<<mark2<<endl;

}

void test :: display()

{

total = mark1+mark2;

cout<<total;

}

int main()

{

details det;

det.getno(101);

det.putno();

test tes;

tes.getmarks(10,20);

tes.putmarks();

tes.display();

return 0;

}