CEN 591 –Object Oriented Programming Lab File

BTech Computer Engineering  
Vth Semester

Submitted by:

Mohammad Kashif (20BCS063)

Submitted to:  
Dr. Wakar Ahmad

*Department of Computer Engineering,*

*Faculty of Engineering & Technology,*

Jamia Millia Islamia, New Delhi2022

Index

|  |  |  |
| --- | --- | --- |
| Sr No | Name of Program | Page No |
| 1. | Write a program to implement STUDENT class consisting of name, enrollId and marks as class data members. Create three objects for the class using the concept of array of objects. Write member functions to read and display the student information. Also write the main program to create objects and call the member functions from the class. | 4-7 |
| 2. | Write a C++ program handling the following details for students and staff using inheritance.  • Student Details: name, address, percentage marks  • Staff Details: name, address, salary  Create appropriate base and derived classes. Input the details and output them. | 8-9 |
| 3. | Write a C++ program to perform the addition of two time objects in hour and minute format, display the result in hour: minute format using object as a function argument. | 10 |
| 4. | Write a C++ program based on following scenario:  Consider an example of a bookshop which sells books and video tapes. These two classes are inherited from the base class called media. The media class has command data members such as title and publication. The book class has data members for storing a number of pages in a book, and the tape class has the playing time in a tape. Each class will have member functions such as read() and show(). In the base class, these members have to be defined as virtual functions. Write a program which models the class hierarchy for the bookshop and processes objects of these classes  using pointers to the base class. | 11-12 |
| 5. | Write a C++ program to overload [] operator for the following scenario:  Create a class AccountBook that contains account holder details such as name and account number. Take input for 5 account holders in the account table. When we enter account number, then the program prints account holder name while entering of account holder name, it prints account number of holder. | 13-15 |
| 6. | Write a C++ Program to implement Complex class representing complex numbers. Include operator functions to overload the operators +=, -=, \*=, /= and the << operator for the class. Here << operator should be used for printing the results of complex number operation. | 16-17 |
| 7. | Design classes such that they support the following statements: Rupee rl, r2; Dollar dl, d2; dl = r2; // converts rupee (Indian currency) to dollar (US currency) r2 = d2; // converts dollar (US currency) to rupee (Indian currency) Write a complete program which does such conversions according to the world market value. | 18-19 |
| 8. | Write suitable C++ program to implement following OOPS concepts:  (a) Pure Virtual Function  (b) Pointers to Derived Class Object  (c) Virtual Destructor  (d) Overloading through friend function | 20-21 |
| 9. | Write a java program to create an abstract class named Shape that contains two integers and an empty method named printArea(). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contain  only the method printArea( ) that prints the area of the given shape. | 22-23 |
| 10. | Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number. | 24-25 |

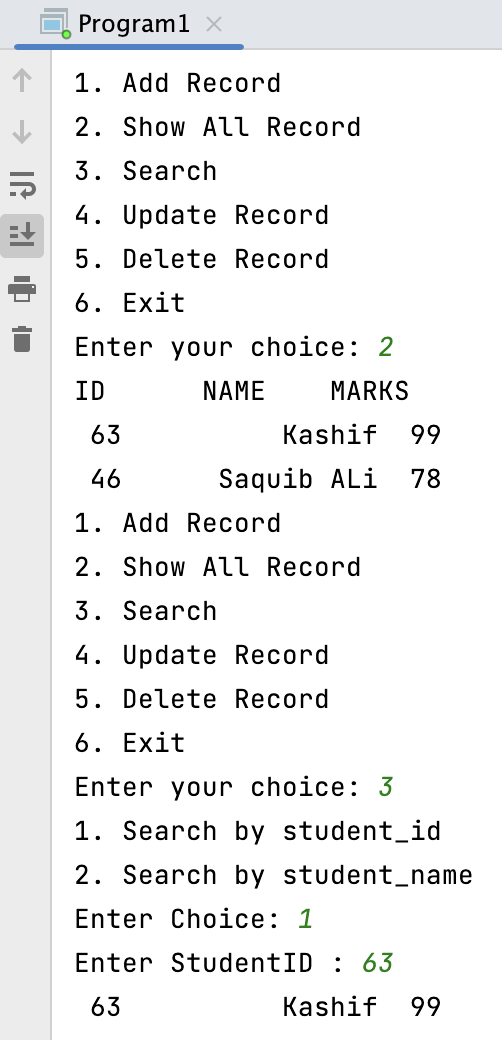
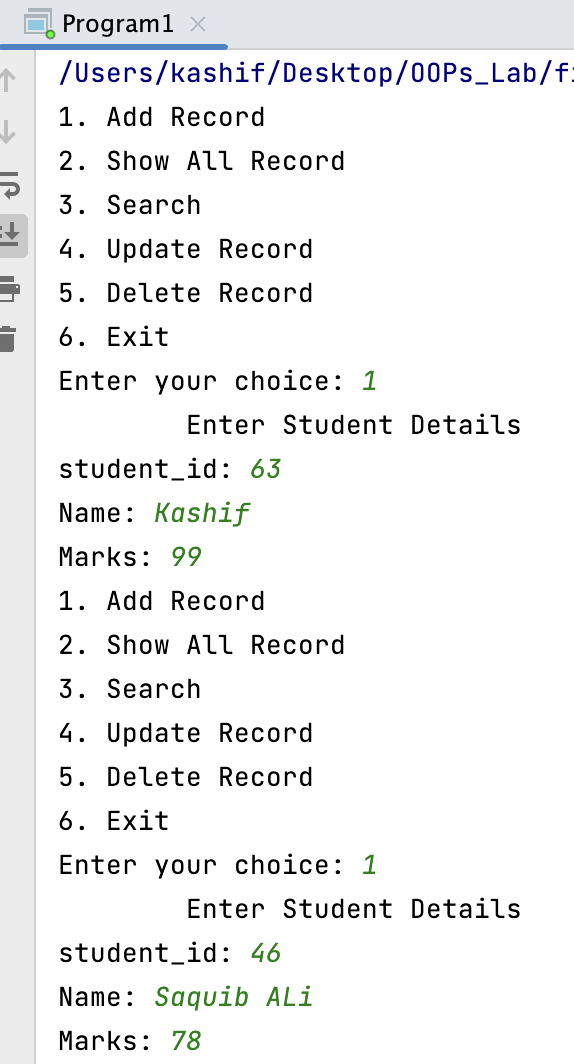
**Program 1: Write a program to implement STUDENT class consisting of name, enrollId and**

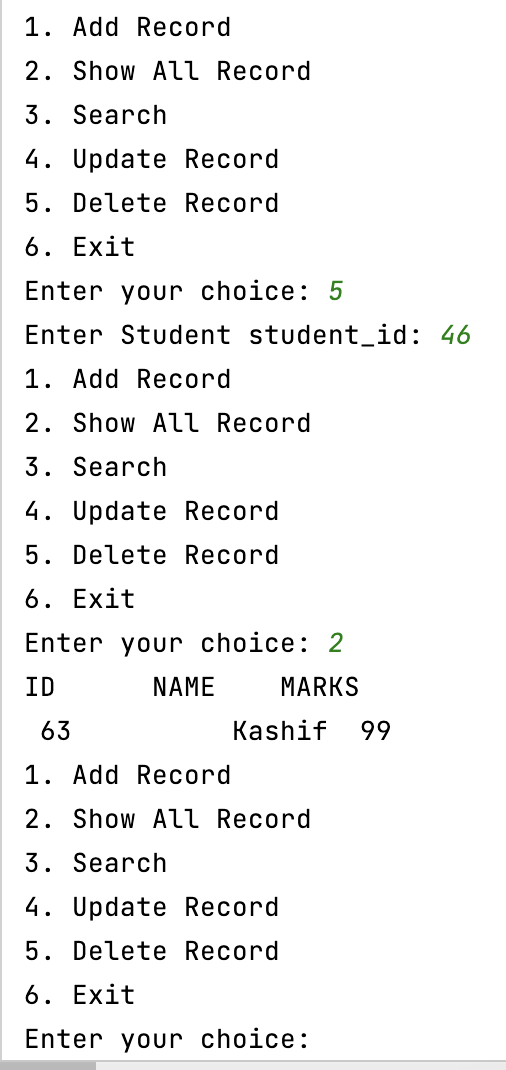
**marks as class data members. Create three objects for the class using the concept of array of objects. Write member functions to read and display the student information. Also write the main program to create objects and call the member functions from the class.**

Code :

#include **<iostream>**#include **<cstring>  
using namespace** std;  
  
**class** Student  
{  
**private**:  
 **int** student\_id;  
 **char** student\_name[30];  
 **int** student\_marks;  
  
**public**:  
 **void** inputData()  
 {  
 cout << **"\t\tEnter Student Details\t\t\n"**;  
 cout << **"student\_id: "**;  
 cin >> student\_id;  
 getchar();  
 cout << **"Name: "**;  
 cin.getline(student\_name, 30);  
 cout << **"Marks: "**;  
 cin >> student\_marks;  
 }  
 **void** displayData()  
 {  
 printf(**"%3d\t%15s\t%3d\n"**, **this**->student\_id, **this**->student\_name, **this**->student\_marks);  
 }  
 **void** update\_Data()  
 {  
 getchar();  
 cout << **"Update Student student\_name ("** << student\_name << **"): "**;  
 cin.getline(student\_name, 30);  
 cout << **"Update Student student\_marks ("** << student\_marks << **"): "**;  
 cin >> student\_marks;  
 }  
 **bool** id\_comparision(**int** id)  
 {  
 **return this**->student\_id == id;  
 }  
 **bool** compare\_name(**char** name[])  
 {  
 **return** !strcmp(**this**->student\_name, name);  
 }  
};  
**int** main()  
{  
  
 Student s[10];  
 **int** count = 0;  
 **while** (1)  
 {  
 **int** ch;  
 cout << **"1. Add Record\n2. Show All Record\n3. Search\n4. Update Record\n5. Delete Record\n6. Exit\n"**;  
 cout << **"Enter your choice: "**;  
 cin >> ch;  
 **switch** (ch)  
 {  
 **case** 1:  
 s[count++].inputData();  
 **break**;  
 **case** 2:  
 **if** (!count)  
 {  
 cout << **"No record found...\n"**;  
 **break**;  
 }  
  
 cout << **"ID\t\tNAME\tMARKS\n"**;  
  
 **for** (**int** i = 0; i < count; ++i)  
 {  
 s[i].displayData();  
 }  
 **break**;  
 **case** 3:  
 **int** choice;  
 cout << **"1. Search by student\_id\n2. Search by student\_name\n"**;  
 cout << **"Enter Choice: "**;  
 cin >> choice;  
 **switch** (choice)  
 {  
 **case** 1:  
 **int** id;  
 cout << **"Enter StudentID : "**;  
 cin >> id;  
 **bool** flag;  
 flag = **true**;  
 **for** (**int** i = 0; i < count; ++i)  
 {  
 **if** (s[i].id\_comparision(id))  
 {  
 flag = **false**;  
 s[i].displayData();  
 }  
 }  
 **if** (flag)  
 cout << **"No Record found with StudentID : "** << id << endl;  
 **break**;  
 **case** 2:  
 **char** name[30];  
 cout << **"Enter Student student\_name: "**;  
 getchar();  
 flag = **true**;  
 cin.getline(name, 30);  
 **for** (**int** i = 0; i < count; ++i)  
 {  
 **if** (s[i].compare\_name(name))  
 {  
 flag = **false**;  
 s[i].displayData();  
 }  
 }  
 **if** (flag)  
 cout << **"No Record found with student\_name : "** << name << endl;  
 **break**;  
 }  
 **break**;  
 **case** 4:  
 cout << **"Enter Student student\_id: "**;  
 **int** id;  
 cin >> id;  
 **for** (**int** i = 0; i < count; ++i)  
 {  
 **if** (s[i].id\_comparision(id))  
 {  
 s[i].update\_Data();  
 }  
 }  
 **break**;  
 **case** 5:  
 cout << **"Enter Student student\_id: "**;  
 cin >> id;  
 **int** index;  
 **bool** flag;  
 flag = **false**;  
 **for** (**int** i = 0; i < count; ++i)  
 {  
 **if** (s[i].id\_comparision(id))  
 {  
 flag = **true**;  
 index = i;  
 }  
 **if** (flag)  
 {  
 **while** (index < count)  
 {  
 s[index] = s[index + 1];  
 index++;  
 }  
 cout << **"Student Record deleted with student\_id : "** << id << endl;  
 }  
  
 count--;  
 }  
 **break**;  
 **case** 6:  
 cout << **"Exiting..."** << endl;  
 exit(1);  
 }  
 }  
 **return** 0;  
}

Output





**Program 2: Write a C++ program handling the following details for students and staff using**

**inheritance.**

**• Student Details: name, address, percentage marks**

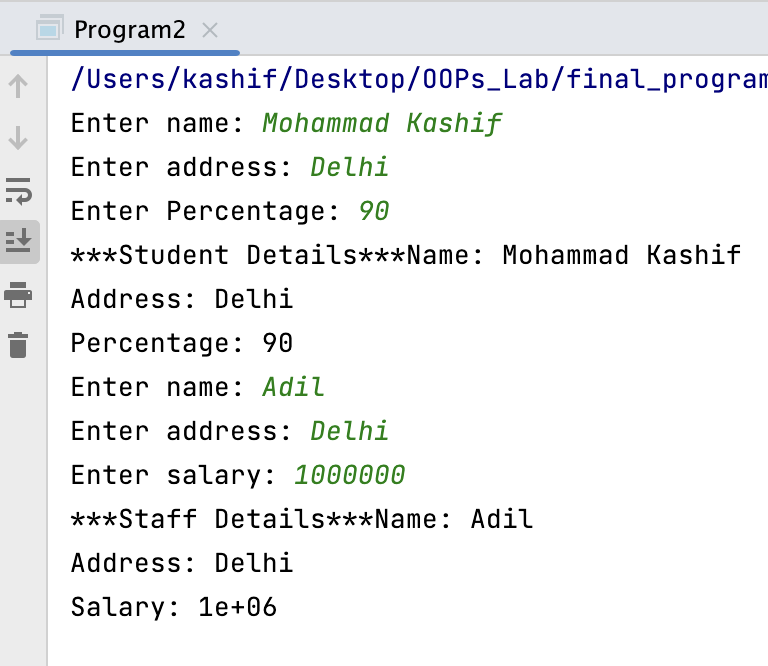
**• Staff Details: name, address, salary**

**Create appropriate base and derived classes. Input the details and output them.**

Code :

#include **<iostream>  
using namespace** std;  
**class** Person  
{  
**protected**:  
 string name;  
 string address;  
  
**public**:  
 Person(string \_name, string \_address)  
 {  
 name = \_name;  
 address = \_address;  
 }  
 **void** getData()  
 {  
 cout << **"Name: "** << name << endl;  
 cout << **"Address: "** << address << endl;  
 }  
};  
  
**class** Student : **public** Person  
{  
**protected**:  
 **float** percentage;  
  
**public**:  
 Student(string \_name, string \_address, **float** \_percentage) : Person(\_name, \_address)  
 {  
 percentage = \_percentage;  
 }  
 **void** getData()  
 {  
 cout << **"\*\*\*Student Details\*\*\*"**;  
 Person::getData();  
 cout << **"Percentage: "** << percentage << endl;  
 }  
};  
  
**class** Staff : **public** Person  
{  
**protected**:  
 **float** salary;  
  
**public**:  
 Staff(string \_name, string \_address, **float** \_salary) : Person(\_name, \_address)  
 {  
 salary = \_salary;  
 }  
 **void** getData()  
 {  
 cout << **"\*\*\*Staff Details\*\*\*"**;  
 Person::getData();  
 cout << **"Salary: "** << salary << endl;  
 }  
};  
  
**int** main()  
{  
 string name, address;  
 cout << **"Enter name: "**;  
 getline(cin, name);  
  
 cout << **"Enter address: "**;  
 getline(cin, address);  
  
 **float** percentage;  
 cout << **"Enter Percentage: "**;  
 cin >> percentage;  
  
 Student s(name, address, percentage);  
 s.getData();  
 getchar();  
  
 cout << **"Enter name: "**;  
 getline(cin, name);  
  
 cout << **"Enter address: "**;  
 getline(cin, address);  
  
 **float** salary;  
 cout << **"Enter salary: "**;  
 cin >> salary;  
  
 Staff st(name, address, salary);  
 st.getData();  
 **return** 0;  
}

Output

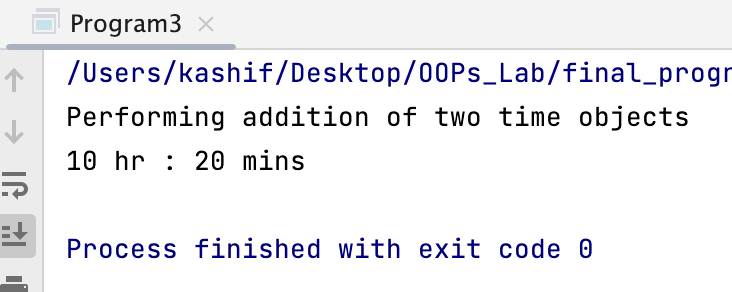


Program 3: Write a C++ program to perform the addition of two time objects in hour and minute format, display the result in hour: minute format using object as a function argument.

Code :

#include **<iostream>  
using namespace** std;  
  
**class** Time{  
**private**:  
 **int** hr , min;  
**public**:  
 Time(**int** hour,**int** minute){  
 hr =hour;  
 min = minute;  
 }  
 **void operator** + (Time obj){  
 cout << **"Performing addition of two time objects"** <<endl;  
 min = min + obj.min;  
 **int** r = min/60;  
 **if**(r)  
 min = min%60;  
 hr = hr + obj.hr+r;  
 }  
 **void** showData() **const**{  
 cout << hr <<**" hr : "**<< min<<**" mins"**<<endl;  
 };  
  
};  
  
**int** main() {  
 Time t1(04,30) ,t2(2,230);  
 t1+t2;  
 t1.showData();  
 **return** 0;  
}

Output :

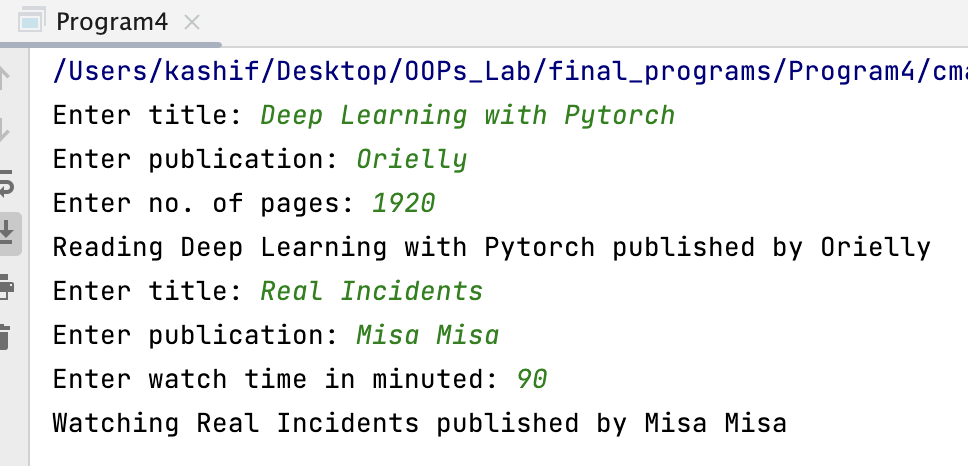


Program 4: Write a C++ program based on following scenario: Consider an example of a bookshop which sells books and video tapes. These two classes are inherited from the base class called media. The media class has command data members such as title and ublication. The book class has data members for storing a number of pages in a book, and the tape class has the playing time in a tape. Each class will have member functions such as read() and show(). In the base class, these members have to be defined as virtual functions. Writea program which models the class hierarchy for the bookshop and processes objects of these classes

using pointers to the base class.

#include **<iostream>  
using namespace** std;  
**class** Media{  
**protected**:  
 string title;  
 string publication;  
**public**:  
 Media(){};  
 Media(string \_title, string \_publication) {  
 title = \_title;  
 publication = \_publication;  
 }  
 **virtual void** read()  
 {  
 cout << **"Reading Media"** << endl;  
 }  
 **virtual void** show()  
 {  
 cout << **"Watching Media"** << endl;  
 }  
 **virtual** ~Media()  
 {  
 cout << **"Media Deleted"** << endl;  
 }  
};  
  
**class** Book : **public** Media  
{  
**protected**:  
 **int** pages;  
  
**public**:  
 Book(string \_title, string \_publication, **int** \_pages) : Media(\_title, \_publication)  
 {  
 pages = \_pages;  
 }  
 **void** read()  
 {  
 cout << **"Reading "** << title << **" published by "** << publication << endl;  
 }  
 ~Book()  
 {  
 cout << **"Book deleted"** << endl;  
 }  
};  
**class** Tape : **public** Media  
{  
**protected**:  
 **int** playing\_time;  
  
**public**:  
 Tape(string \_title, string \_publication, **int** \_playing\_time) : Media(\_title, \_publication)  
 {  
 playing\_time = \_playing\_time;  
 }  
 **void** show()  
 {  
 cout << **"Watching "** << title << **" published by "** << publication << endl;  
 }  
 ~Tape()  
 {  
 cout << **"Tape Deleted"** << endl;  
 }  
};  
  
**int** main()  
{  
 Media \*media;  
 string title, publication;  
 **int** pages, time;  
 cout << **"Enter title: "**;  
 getline(cin, title);  
 *// getchar();* cout << **"Enter publication: "**;  
 getline(cin, publication);  
 cout << **"Enter no. of pages: "**;  
 cin >> pages;  
 media = **new** Book(title, publication, pages);  
 media->read();  
 getchar();  
 cout << **"Enter title: "**;  
 getline(cin, title);  
 cout << **"Enter publication: "**;  
 getline(cin, publication);  
 cout << **"Enter watch time in minuted: "**;  
 cin >> time;  
 media = **new** Tape(title, publication, time);  
 media->show();  
 **return** 0;  
}

Output



**Program 5: Write a C++ program to overload [] operator for the following scenario:**

**Create a class AccountBook that contains account holder details such as name and account number. Take input for 5 account holders in the account table. When we enter account number, then the program prints account holder name while entering of account holder name, it prints account number of holder**.

Code :

#include **<iostream>**#include **<vector>  
using namespace** std;  
**class** Account\_holder  
{

**private**:  
 string name;  
 string account\_number;  
**public**:  
 Account\_holder(){};  
 Account\_holder(string \_name, string \_account\_number)  
 {

name = \_name;  
 account\_number = \_account\_number;  
 }  
 **friend bool operator**==(string s, Account\_holder a);  
 **friend** ostream &**operator**<<(ostream &out, Account\_holder a);  
};  
**bool operator**==(string s, Account\_holder a)  
{

**return** (s == a.account\_number **or** s == a.name);  
}  
ostream &**operator**<<(ostream &out, Account\_holder a)  
{

out << **"Name: "** << a.name << endl;  
 out << **"Account Number: "** << a.account\_number << endl;  
 **return** out;  
}  
**class** AccountBook{  
**private**:  
 vector<Account\_holder> account\_table;  
**public**:  
 AccountBook(){};  
 **void** add\_account(Account\_holder a)  
 {  
 account\_table.push\_back(a);  
 }  
 Account\_holder **operator**[](string s)  
 {  
 **for** (**int** i = 0; i < account\_table.size(); i++)  
 {  
 **if** (s == account\_table[i])  
 {  
 **return** account\_table[i];  
 }  
 }  
 **return** Account\_holder();  
 }  
};  
  
**int** main()  
{  
 AccountBook ab;  
 **int** choice;  
 string name, account\_number;  
 **while** (**true**)  
 {  
 cout << **"1. Add Account"** << endl;  
 cout << **"2. Search Account"** << endl;  
 cout << **"3. Exit"** << endl;  
 cout << **"Enter your choice: "**;  
 cin >> choice;  
 **switch** (choice)  
 {  
 **case** 1:  
 cout << **"Enter name: "**;  
 getchar();  
 getline(cin, name);  
 cout << **"Enter account number: "**;  
 getline(cin, account\_number);  
 ab.add\_account(Account\_holder(name, account\_number));  
 **break**;  
 **case** 2:  
 cout << **"Enter name or account number: "**;  
 getchar();  
 getline(cin, name);  
 cout << ab[name];  
 **break**;  
 **case** 3:  
 exit(0);  
 **default**:  
 cout << **"Invalid choice"** << endl;  
 }  
 }  
 **return** 0;  
}

Output

A picture containing text

Description automatically generated

**Program 6: Write a C++ Program to implement Complex class representing complex numbers. A complex number in mathematics is defined as x + iy where x defines the real part of the number and y is the imaginary part. The letter i represents the square root of -1 (which means i2 is -1). Include operator functions to overload the operators +=, -=, \*=, /= and the << operator for the class. Here << operator should be used for printing the results of complex number operation.**

Code

#include **<iostream>  
using namespace** std;  
  
**class** Complex  
{  
**private**:  
 **int** real;  
 **int** img;  
  
**public**:  
 Complex(){};  
 Complex(**int** \_real, **int** \_img);  
 Complex **operator**+=(Complex m)  
 {  
 real += m.real;  
 img += m.img;  
 **return** \***this**;  
 }  
 Complex **operator**-=(Complex m)  
 {  
 real -= m.real;  
 img -= m.img;  
 **return** \***this**;  
 }  
 Complex **operator**\*=(Complex m)  
 {  
 real = real \* m.real - img \* m.img;  
 img = real \* m.img + img \* m.real;  
 **return** \***this**;  
 }  
 Complex **operator**/=(Complex m)  
 {  
 real = (real \* m.real + img \* m.img) / (m.real \* m.real + m.img \* m.img);  
 img = (img \* m.real - real \* m.img) / (m.real \* m.real + m.img \* m.img);  
 **return** \***this**;  
 }  
 **friend** ostream &**operator**<<(ostream &out, Complex &c);  
};  
  
Complex::Complex(**int** \_real, **int** \_img)  
{  
 real = \_real;  
 img = \_img;  
}  
  
ostream &**operator**<<(ostream &out, Complex &c)  
{  
 out << c.real << **" + "** << c.img << **"i"** << endl;  
 **return** out;  
}  
**int** main()  
{  
 Complex c1(1, 2), c2(3, 4);  
 cout << **"Complex C1: "** << c1;  
 cout << **"Complex C2: "** << c2;  
 c1 += c2;  
 cout << **"Complex C1 after c1 += c2: "** << c1;  
 c1 -= c2;  
 cout << **"Complex C1 after c1 -= c2: "** << c1;  
 c1 \*= c2;  
 cout << **"Complex C1 after c1 \*= c2: "** << c1;  
 c1 /= c2;  
 cout << **"Complex C1 after c1 /= c2: "** << c1;  
 **return** 0;  
}

Output :

Text

Description automatically generated

**Program 7: Design classes such that they support the following statements: Rupee rl, r2; Dollar dl, d2; dl = r2; // converts rupee (Indian currency) to dollar (US currency) r2 = d2; // converts dollar (US currency) to rupee (Indian currency) Write a complete program which does such conversions according to the world market value.**

Code :

#include **<iostream>  
using namespace** std;  
  
**class** Dollar  
{  
**public**:  
 **float** amount;  
 Dollar(){};  
 Dollar(**float** \_amount) *// normal amount in dollar* {  
 amount = \_amount;  
 }  
 **friend** ostream &**operator**<<(ostream &out, Dollar &d); *// to print*};  
**class** Rupee  
{  
**public**:  
 **float** amount;  
 Rupee(){};  
 Rupee(**float** \_amount) *// normal amount in rupee* {  
 amount = \_amount;  
 }  
 **operator** Dollar() *// case 1 : type conversion from rupee to dollar d1 = r1 conversion in source class* {  
 Dollar d;  
 d.amount = amount / 82.32;  
 **return** d;  
 }  
 Rupee(Dollar &d) *// case 2: type conversion from dollor to rupee r2 = d2* {  
 amount = 82.32 \* d.amount;  
 }  
 **friend** ostream &**operator**<<(ostream &out, Rupee &r); *// to print*};  
ostream &**operator**<<(ostream &out, Rupee &r)  
{  
 out << r.amount;  
 **return** out;  
}  
  
ostream &**operator**<<(ostream &out, Dollar &d)  
{  
 out << d.amount;  
 **return** out;  
}  
  
**int** main()  
{  
 **float** amount;  
 cout << **"Enter amount in Rupees : "**;  
 cin >> amount;  
 Rupee r1(amount); *// rupee with given amount* Dollar d1;  
 d1 = r1; *// destination d1 and source r1* cout << r1.amount << **" Rupees in Dollars -> "** << d1 << endl;  
 cout << **"Enter amount in Dollar : "**;  
 cin >> amount;  
 Dollar d2(amount); *// dollar with given amount* Rupee r2;  
 r2 = d2; *// destination r2 and source d2* cout << d2.amount << **" Dollar in Rupees -> "** << r2 << endl;  
 **return** 0;  
}

Output

Text

Description automatically generated

**Program 8: Write suitable C++ program to implement following OOPS concepts:**

**(a) Pure Virtual Function**

**(b) Pointers to Derived Class Object**

**(c) Virtual Destructor**

**(d) Overloading through friend function**

Code :

#include **<iostream>  
using namespace** std;  
  
**class** base  
{  
**public**:  
 base()  
 {  
 cout << **"Base class constructor"** << endl;  
 }  
 *// Pure Virtual Function* **virtual void** display() = 0;  
 *// virtual destructor* **virtual** ~base()  
 {  
 cout << **"Destructor of base class"** << endl;  
 }  
};  
  
**class** derived : **public** base  
{  
**public**:  
 derived()  
 {  
 cout << **"Derived class constructor"** << endl;  
 }  
 **void** display()  
 {  
 cout << **"Derived class display function"** << endl;  
 }  
  
 *// destructor* ~derived()  
 {  
 cout << **"Derived class destructor"** << endl;  
 }  
};  
  
*// overloading through friend function***class** complex  
{  
 **int** a, b;  
  
**public**:  
 complex(**int** x, **int** y)  
 {  
 a = x;  
 b = y;  
 }  
 **friend** complex **operator**+(complex, complex);  
 **friend** ostream &**operator**<<(ostream &, complex);  
};  
  
complex **operator**+(complex c1, complex c2)  
{  
 complex temp(0, 0);  
 temp.a = c1.a + c2.a;  
 temp.b = c1.b + c2.b;  
 **return** temp;  
}  
  
ostream &**operator**<<(ostream &out, complex c)  
{  
 out << c.a << **" + "** << c.b << **"i"** << endl;  
 **return** out;  
}  
**int** main()  
{  
 base \*b;  
 derived d;  
 b = &d;  
 b->display();  
 complex c1(2, 3), c2(4, 5), c3(0, 0);  
 c3 = c1 + c2;  
 cout << c3;  
 **return** 0;  
}

Output :

Graphical user interface, text, application

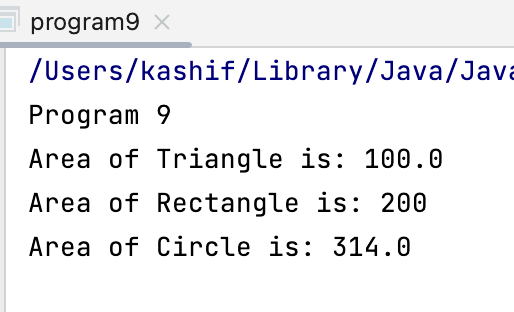
Description automatically generated with medium confidence

**Program 9: Write a java program to create an abstract class named Shape that contains two integers and an empty method named printArea(). Provide three classes named Rectangle, Triangle and Circle such that each one of the classes extends the class Shape. Each one of the classes contain only the method printArea( ) that prints the area of the given shape.**

Code :

**abstract class** Shape {  
 **int param1**;  
 **int param2**;  
  
 **abstract void** printArea();  
}  
  
**class** Triangle **extends** Shape {  
 Triangle(**int** \_height, **int** \_base) {  
 param1 = \_height;  
 param2 = \_base;  
 }  
  
 **void** printArea() {  
 System.***out***.println(**"Area of Triangle is: "** + (0.5 \* param1 \* param2));  
 }  
}  
  
**class** Rectangle **extends** Shape {  
 Rectangle(**int** \_length, **int** \_breadth) {  
 param1 = \_length;  
 param2 = \_breadth;  
 }  
  
 **void** printArea() {  
 System.***out***.println(**"Area of Rectangle is: "** + (param1 \* param2));  
 }  
}  
  
**class** Circle **extends** Shape {  
 Circle(**int** \_radius) {  
 param1 = \_radius;  
 }  
  
 **void** printArea() {  
 System.***out***.println(**"Area of Circle is: "** + (3.14 \* param1 \* param1));  
 }  
}  
  
**public class** program9 {  
  
 **public static void** main(String[] args) {  
 System.***out***.println(**"Program 9"**);  
 Triangle t = **new** Triangle(10, 20);  
 Rectangle r = **new** Rectangle(10, 20);  
 Circle c = **new** Circle(10);  
 t.printArea();  
 r.printArea();  
 c.printArea();  
 }  
}

Output :



Program 10: Write a java program that implements a multi-thread application that has three

threads. First thread generates random integer every 1 second and if the value is even,

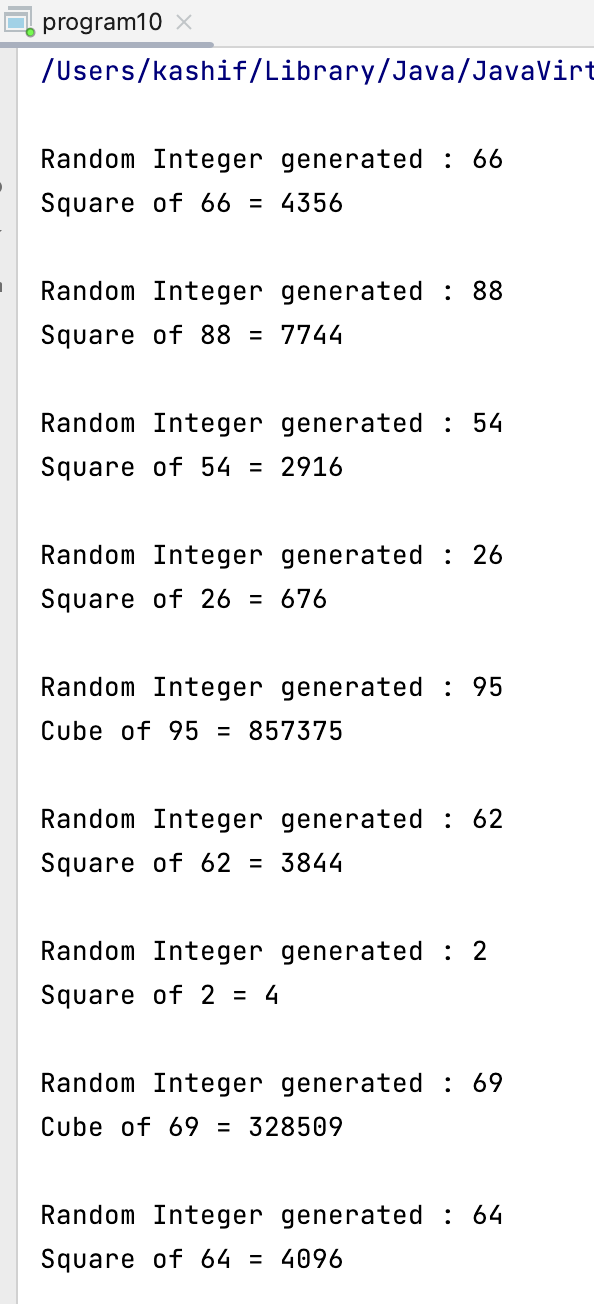
second thread computes the square of the number and prints. If the value is odd, the third

thread will print the value of cube of the number.

Code :

**import** java.util.Random;  
  
**class** Square **extends** Thread {  
 **int x**;  
  
 Square(**int** n) {  
 **x** = n;  
 }  
  
 **public void** run() {  
 **int** sqr = **x** \* **x**;  
 System.***out***.println(**"Square of "** + **x** + **" = "** + sqr);  
 }  
}  
  
**class** Cube **extends** Thread {  
 **int x**;  
  
 Cube(**int** n) {  
 **x** = n;  
 }  
  
 **public void** run() {  
 **int** cub = **x** \* **x** \* **x**;  
 System.***out***.println(**"Cube of "** + **x** + **" = "** + cub);  
 }  
  
}  
  
**class** Number **extends** Thread {  
  
 **public void** run() {  
 Random random = **new** Random();  
  
 **while** (**true**) {  
 **int** randomInteger = random.nextInt(100);  
 System.***out***.println(**"\nRandom Integer generated : "** + randomInteger);  
  
 **if** (randomInteger % 2 == 0) {  
 Square square = **new** Square(randomInteger);  
 square.start();  
 } **else** {  
 Cube cube = **new** Cube(randomInteger);  
 cube.start();  
 }  
  
 **try** {  
 Thread.*sleep*(1000);  
 } **catch** (InterruptedException ex) {  
 System.***out***.println(ex);  
 }  
 }  
 }  
}  
  
**public class** program10 {  
 **public static void** main(String args[]) {  
 Number n = **new** Number();  
 n.start();  
 }  
}

Output :

 Text

Description automatically generated with medium confidence