Program No. 1

Given the coefficients of the quadratic polynomial (float variables), write a C++ program to determine whether the roots are real or complex (imaginary). If the roots are real, find them otherwise write the message “No real roots”.

```
/* C++ program to find roots of a quadratic equation */
#include <bits/stdc++.h>
using namespace std;

// Prints roots of quadratic equation ax*2 + bx + x
void findRoots(int a, int b, int c)
{
    // If a is 0, then equation is not quadratic, but
    // linear
    if (a == 0) {
        cout << "Invalid";
        return;
    }

    int d = b * b - 4 * a * c;
    double sqrt_val = sqrt(abs(d));

    if (d > 0) {
        cout << "Roots are real and different \n";
        cout << (double)(-b + sqrt_val) / (2 * a) << "\n"
             << (double)(-b - sqrt_val) / (2 * a);
    }
    else if (d == 0) {
        cout << "Roots are real and same \n";
        cout << -(double)b / (2 * a);
    }
    else // d < 0
    {
        cout << "Roots are complex \n";
        cout << -(double)b / (2 * a) << " + i" << sqrt_val
             << "\n"
             << -(double)b / (2 * a) << " - i" << sqrt_val;
    }
}

// Driver code
int main()
{
    int a = 1, b = -7, c = 12;

    // Function call
    findRoots(a, b, c);
    return 0;
}
```

Program No. 2

An electricity board charges the following rates to domestic users to discourage large consumption of energy
For the first 100 units:- 60 P per unit For the next 200 units:-
80 P per unit Beyond 300 units:-90 P per unit
All users are charged a minimum of Rs 50 if the total amount is more than Rs 300 then an additional surcharge of 15% is added. WAP to read the names of users and number of units consumed and display the charges with names.

```
#include<iostream>

using namespace std;

int main()

{

int no_unit;

float charge,scharge;

char name[20];

cout<< "\nEnter name and number of units consumed";

cin>>name;

cin>>no_unit;

if(no_unit<=100)

charge=(0.60*no_unit);

else if(no_unit>100&&no_unit<=300)

{

charge=(100*0.60);

charge+=((no_unit-100)*0.80);

}

else if(no_unit>=300)

{

charge=(100*0.60);

charge+=(200*0.80);

charge+=((no_unit-300)*0.90);

}

if(charge<50)

charge=50;

if(charge>300)
{
scharge=(0.15*charge);
charge+=scharge;
}
cout<< "electricity bill \n";
cout<<name<<" : : rs"<<charge;
return(0);
}
```

Program No. 3

W.A.P in C++ by defining a class to represent a bank account. Include the following - Data Members

- **Name of the depositor**
- **Account number**
- **Type of account (Saving, Current etc.)**
- **Balance amount in the account**

Member Functions

- **To assign initial values**
- **To deposit an amount**
- **To withdraw an amount after checking the balance**
- **To display name and balance**

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

using namespace std;

class bank
{
        int acno;
        char nm[100], acctype[100];
        float bal;
   public:
        bank(int acc_no, char *name, char *acc_type, float balance)  //Parameterized Constructor
        {
                acno=acc_no;
                strcpy(nm, name);
                strcpy(acctype, acc_type);
                bal=balance;
        }
        void deposit();
```

```
    void withdraw();
    void display();
};
void bank::deposit()   //depositing an amount
{
    int damt1;
    cout<<"\n Enter Deposit Amount = ";
    cin>>damt1;
    bal+=damt1;
}
void bank::withdraw()  //withdrawing an amount
{
    int wamt1;
    cout<<"\n Enter Withdraw Amount = ";
    cin>>wamt1;
    if(wamt1>bal)
        cout<<"\n Cannot Withdraw Amount";
    bal-=wamt1;
}
void bank::display()  //displaying the details
{
    cout<<"\n ----------------------";
    cout<<"\n Accout No. : "<<acno;
    cout<<"\n Name : "<<nm;
    cout<<"\n Account Type : "<<acctype;
    cout<<"\n Balance : "<<bal;
}
int main()
{
    int acc_no;
```

```
        char name[100], acc_type[100];
        float balance;
        cout<<"\n Enter Details: \n";
        cout<<"-----------------------";
        cout<<"\n Accout No. ";
        cin>>acc_no;
        cout<<"\n Name : ";
        cin>>name;
        cout<<"\n Account Type : ";
        cin>>acc_type;
        cout<<"\n Balance : ";
        cin>>balance;

        bank b1(acc_no, name, acc_type, balance);  //object is created
        b1.deposit(); //
        b1.withdraw(); // calling member functions
        b1.display(); //
        return 0;
}
```

Program No. 4

W.A.P in C++ to show the working of function overloading by using a function named calculateArea() to calculate area of square, rectangle and triangle using different signatures as required.

```
/* C++ program to find Area using Function Overloading  */


#include<iostream>

using namespace std;

int area(int);

int area(int,int);

float area(float);

float area(float,float);

int main()

{

        int s,l,b;

        float r,bs,ht;

        cout<<"Enter side of a square:";

        cin>>s;

        cout<<"Enter length and breadth of rectangle:";

        cin>>l>>b;

        cout<<"Enter radius of circle:";

        cin>>r;

        cout<<"Enter base and height of triangle:";

        cin>>bs>>ht;

        cout<<"Area of square is"<<area(s);

        cout<<"\nArea of rectangle is "<<area(l,b);

    cout<<"\nArea of circle is "<<area(r);

    cout<<"\nArea of triangle is "<<area(bs,ht);

}

int area(int s)

{
    return(s*s);
}
int area(int l,int b)
{
    return(l*b);
}
float area(float r)
{
    return(3.14*r*r);
}
float area(float bs,float ht)
{
   return((bs*ht)/2);
}
```

Program No. 5

Write a Program in C++ to demosntrate the concept of data abstraction using the concept of Class and Objects

```
#include <iostream>

using namespace std;


class implementAbstraction

{

        private:

                int a, b;


        public:


                // method to set values of

                // private members

                void set(int x, int y)

                {

                        a = x;

                        b = y;

                }


                void display()

                {

                        cout<<"a = " <<a << endl;

                        cout<<"b = " << b << endl;

                }

};


int main()

{

        implementAbstraction obj;
```

```
    obj.set(10, 20);
    obj.display();
    return 0;
}
```

Program No. 6

Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance

Data Members -

- **partNumber (type String)**
- **partDescription (type String)**
- **quantity of the item being purchased (type int) price_per_item (type double)**

Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount() that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named invoiceTest that demonstrates class Invoice's capabilities.

```
#include<iostream>

#include <string>

using namespace std;


class Invoice

{

 public:

  Invoice( string, string, int, int );

  void setPartNumber( string );

  string getPartNumber();

  void setPartDescription(string);

  string getPartDescription();

  void setItemQuantity(int);

  int getItemQuantity();

  void setItemPrice(int);

  int getItemPrice();

  int getInvoiceAmount();

 private:

  string partNumber;

  string partDescription;
  int itemQuantity;
  int itemPrice;
};
Invoice::Invoice( string number, string description, int quantity, int price )
{
 partNumber=number;
 partDescription=description;
 if(quantity>0)
  itemQuantity=quantity;
 else
 {
  itemQuantity=0;
  cout<<"Initial quantity was invalid."<<endl;
 }
 if(price>0)
  itemPrice=price;
 else
 {
  itemPrice=0;
  cout<<"Initial price was invalid."<<endl;
 }
}
void Invoice::setPartNumber( string number)
{
 if ( number.length() <= 25 )
  partNumber = number;
 if ( number.length() > 25 )
 {
  partNumber = number.substr( 0, 25 );
  cout << "Name \"" << number <<"\" exceeds maximum length (25).\n"<< "Limiting
partNumber to first 25 characters.\n" << endl;
```

```
 }
}
void Invoice::setPartDescription(string description )
{
 if ( description.length() <= 25 )
  partDescription = description;
 if ( description.length() > 25 )
 {
  partDescription = description.substr( 0, 25 );
  cout << "Name \"" << description <<"\" exceeds maximum length (25).\n"<< "Limiting
partDescription to first 25 characters.\n" << endl;
 }
}
void Invoice::setItemQuantity(int quantity )
{
 if(quantity>0)
  itemQuantity=quantity;
 else
 {
  itemQuantity=0;
  cout<<"Initial quantity was invalid."<<endl;
 }
}
void Invoice::setItemPrice(int price )
{
 if(price>0)
  itemPrice=price;
 else
 {
  itemPrice=0;
  cout<<"Initial price was invalid."<<endl;
 }
}
string Invoice::getPartNumber()
{
 return partNumber;
}
string Invoice::getPartDescription()
{
 return partDescription;
}
int Invoice::getItemQuantity()
{
 return itemQuantity;
}
int Invoice::getItemPrice()
{
 return itemPrice;
}
int Invoice::getInvoiceAmount()
{
 return itemQuantity*itemPrice;
}
int main()
{
 Invoice Invoice1("ed34","Screw Guage",2,30);
 Invoice Invoice2("e322","Screws",10,3);
 cout << "Invoice1's initial part number is: "<< Invoice1.getPartNumber()<< "\nand part
description is: "<< Invoice1.getPartDescription()<<endl;
 cout<< "quantity per item is: "<< Invoice1.getItemQuantity()<< "\nprice per item is: "<<
Invoice1.getItemPrice()<< endl;
 cout<<"Invoice1's total amount is: "<<Invoice1.getInvoiceAmount()<<endl<<endl;
 cout << "Invoice2's initial part number is: "<< Invoice2.getPartNumber()<< "\nand part description is: "<< Invoice2.getPartDescription()<<endl;

 cout<< "quantity per item is: "<< Invoice2.getItemQuantity()<< "\nprice per item is: "<< Invoice2.getItemPrice()<< endl;

 cout<<"Invoice2's total amount is: "<<Invoice2.getInvoiceAmount()<<endl;

}
```

Program No. 7

Imagine a tollbooth with a class called TollBooth. The two data items are of type unsigned int and double to hold the total number of cars and total amount of money collected. A constructor initializes both of these data members to 0. A member function called payingCar( ) increments the car total and adds 0.5 to the cash total. Another function called nonPayCar( ) increments the car total but adds nothing to the cash total. Finally a member function called display( ) shows the two totals. Include a program to test this class. This program should allow the user to push one key to count a paying car , and another to count a non paying car. Pushing the ESC key should cause the program to print out the total number of cars and total cash and then exit.

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

class tollbooth
{
    unsigned int car;
    double amt;
    public:
        tollbooth()
        {
            this->car = 0;
            this->amt = 0;
        }
        void payingcar()
        {
            this->car++;
            this->amt+=0.50;
        }
        void nonpayingcar()
        {
            this->car++;
        }
        void display()
        {
            cout<<"Number of cars: "<<car<<endl;
            cout<<"Amount: "<<amt<<endl;
        }
};

int main()
{
    char c='y';
    int ch;
    tollbooth t;
    do{
    cout<<" 1 for paying \n 2 for nopaying \n 3 Display/Exit \n";
    cout<<"Enter choice \n";
    cin>>ch;
    switch(ch)
    {
    case 1:
    t.payingcar();
    cout<<"car added";
    break;
    case 2:
    t.nonpayingcar();
    cout<<"car added";
    break;
    case 3:
    t.display();
    c='n';
    break;
    }
   // t.display();
   if(c=='y'||c=='Y'){
    cout<<"\nDo you want to continue";
    cin>>c;
   }
    }
    while(c=='y'||c=='Y');
    return 0;
}
```

Program No. 8

Create a class called Time that has separate int member data for hours, minutes and seconds. One constructor should initialize this data to 0, and another should initialize it to fixed values. A member function should display it in 11:59:59 format. A member function named add() should add two objects of type time passed as arguments. A main ( ) program should create two initialized values together, leaving the result in the third time variable. Finally it should display the value of this third variable.

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

class Time {
private:
int hours;
int minutes;
int seconds;
public:
Time(){
this->hours = 0;
this->minutes = 0;
this->seconds = 0;
};
Time(int hours, int minutes, int seconds) {
this->hours = hours;
this->minutes = minutes;
this->seconds = seconds;
};
int getHours(){
return this->hours;
};
int getMinutes(){
return this->minutes;
};
```

```
int getSeconds() {
return this->seconds;
};
void display(){
cout << hours << ":" << minutes << ":" << seconds << endl;
};
Time add(Time time1, Time time2) {
int hoursAdd = time1.getHours() + time2.getHours();
if (hoursAdd > 23) {
hoursAdd -= 24;
}
int minutesAdd = time1.getMinutes() + time2.getMinutes();
if (minutesAdd > 59) {
minutesAdd -= 60;
hoursAdd += 1;
}
int secondsAdd = time1.getSeconds() + time2.getSeconds();
if (secondsAdd > 59) {
secondsAdd -= 60;
minutesAdd += 1;
}

Time time3(hoursAdd, minutesAdd, secondsAdd);
return time3;
};
};
int main() {
Time time1(12, 40, 30);
Time time2(21, 23, 43);
Time time3;
```

```
time3 = time3.add(time1, time2);
time1.display();
time2.display();
time3.display();
return 0;
}
```

Program No. 9

Create class SavingsAccount. Use a static variable annualInterestRate to store the annual interest rate for all account holders. Each object of the class contains a private instance variable savingsBalance indicating the amount the saver currently has on deposit. Provide method calculateMonthlyInterest to calculate the monthly interest by multiplying the savingsBalance by annualInterestRate divided by 12.This interest should be added to savingsBalance. Provide a static method modifyInterestRate that sets the annualInterestRate to a new value. Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of $2000.00 and $3000.00, respectively. Set annualInterestRate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month's interest and print the new balances for both savers.

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

class SavingsAccount
{
	public:
		SavingsAccount(){}
		SavingsAccount(int value);
		static float annualInterestRate;
		void calculateMonthlyInterest();
		static void modifyIntererestRate(float value);
		float GetBalance() const { return savingsBalance; }
	private:
		float savingsBalance;
};

SavingsAccount::SavingsAccount(int value)
{
	savingsBalance = value;
}

float SavingsAccount::annualInterestRate = 0;
```

```
void SavingsAccount::calculateMonthlyInterest()
{
	savingsBalance = (savingsBalance+(savingsBalance * annualInterestRate) / 12);
}

void SavingsAccount::modifyIntererestRate(float value)
{
	annualInterestRate = value;
}

int main()
{
	SavingsAccount saver1(2000.00);
	SavingsAccount saver2(3000.00);

	SavingsAccount::modifyIntererestRate(4);

	saver1.calculateMonthlyInterest();
	cout << "Saver 1 Savings Balance: $" << saver1.GetBalance() << endl;
	saver2.calculateMonthlyInterest();
	cout << "Saver 2 Savings Balance: $" << saver2.GetBalance() << endl;

	cout << endl;

	SavingsAccount::modifyIntererestRate(5);

	saver1.calculateMonthlyInterest();
	cout << "Saver 1 Savings Balance: $" << saver1.GetBalance() << endl;
	saver2.calculateMonthlyInterest();
```

```
        cout << "Saver 2 Savings Balance: $" << saver2.GetBalance() << endl;


        cout << endl;
        return 0;
}
```

Program No. 10

Create a class Complex having two int type variable named real & img denoting real and imaginary part respectively of a complex number. Overload + , - , == operator to add, to subtract and to compare two complex numbers being denoted by the two complex type objects.

```
// C++ Program to Add Two Complex Numbers


// Importing all libraries
#include<bits/stdc++.h>
using namespace std;


// User Defined Complex class
class Complex {


        // Declaring variables
        public:
                int real, imaginary;


        // Constructor to accept
        // real and imaginary part
        Complex(int tempReal = 0, int tempImaginary = 0)
        {
                real = tempReal;
                imaginary = tempImaginary;
        }


        // Defining addComp() method
        // for adding two complex number
        Complex addComp(Complex C1, Complex C2)
        {
```

```
        // creating temporary variable
        Complex temp;

        // adding real part of complex numbers
        temp.real = C1.real + C2.real;

        // adding Imaginary part of complex numbers
        temp.imaginary = C1.imaginary + C2.imaginary;

        // returning the sum
        return temp;
    }

    // Defining subComp() method
    // for subtracting two complex number

    Complex subComp(Complex C1, Complex C2)
    {
        // creating temporary variable
        Complex temp;

        // adding real part of complex numbers
        temp.real = C1.real - C2.real;

        // adding Imaginary part of complex numbers
        temp.imaginary = C1.imaginary - C2.imaginary;

        // returning the sum
        return temp;
    }
```

```
};

// Main Class
int main()
{

        // First Complex number
        Complex C1(3, 2);

        // printing first complex number
        cout<<"Complex number 1 : "<< C1.real
                                << " + i"<< C1.imaginary<<endl;

        // Second Complex number
        Complex C2(9, 5);

        // printing second complex number
        cout<<"Complex number 2 : "<< C2.real
                                << " + i"<< C2.imaginary<<endl;

        // for Storing the sum
        Complex C3;

        // calling addComp() method
        C3 = C3.addComp(C1, C2);

        // printing the sum
        cout<<"Sum of complex number : "
                                        << C3.real << " + i"
                                        << C3.imaginary;
        // for Storing the sub
        Complex C4;

        // calling subComp() method
        C4 = C4.subComp(C1, C2);

        // printing the sum
        cout<<"\nSub of complex number : "
                                        << C4.real << " + i"
                                        << C4.imaginary;
}
```

Program No. 11

Using the concept of operator overloading.Write a program to overload using with and without friend Function.

a. **Unary –**
b. **Unary ++ preincrement, postincrement**
c. **Unary -- predecrement, postdecrement**

With Friend Function

```
#include<iostream>
using namespace std;
class UnaryFriend
{
    int a=10;
    int b=20;
    public:
        void getvalues()
        {
            cout<<"Values of A  &  B\n";
            cout<<a<<"\n"<<b<<"\n"<<endl;
        }
        void show()
        {
            cout<<a<<"\n"<<b<<"\n"<<endl;
        }
        void friend operator-(UnaryFriend &x);
        void friend operator ++(UnaryFriend &x);
        void friend operator --(UnaryFriend &x);
};
void operator-(UnaryFriend &x)
{
    x.a = -x.a;
    x.b = -x.b;
}
  void operator++(UnaryFriend &x)
    {
       x.a=++x.a;
       x.b=x.b++;
    }
      void operator--(UnaryFriend &x)
    {
       x.a=--x.a;
```

```
        x.b=x.b--;
    }


int main()
{
     UnaryFriend x1;
     UnaryFriend x2;
     UnaryFriend x3;
     x1.getvalues();
     cout<<"Before Overloading\n";
     x1.show();
     cout<<"After Overloading \n";
     -x1;
      x1.show();
     x2.getvalues();
     cout<<"Before Pre and Post increment Overloading\n";
     x2.show();
     cout<<"After Pre and Post increment Overloading \n";
     ++x2;
     x2.show();
     x3.getvalues();
     cout<<"Before Pre and Post decrement Overloading\n";
     x3.show();
     cout<<"After Pre and Post decrement Overloading \n";
     --x3;
     x3.show();
      return 0;
}
```

Without Friend Function

```
// C++ program to demonstrate working of unary increment
// and decrement operators
#include <iostream>
using namespace std;

int main()
{
  // Unary Minus
        int a = 1;
        cout << "a value before Unary Minus: " << a << endl;
        int b = -a;
        cout << "b value after -a : " << b << endl;

        // Post increment
         a = 1;
        cout << "a value Post increment: " << a << endl;
         b = a++;
        cout << "b value after a++ : " << b << endl; //1
        cout << "a value after a++ : " << a << endl; //2

        // Pre increment
        a = 1;
        cout << "a value Pre increment:" << a << endl; //1
        b = ++a;
        cout << "b value after ++a : " << b << endl; //2
        cout << "a value after ++a : "<< a << endl; //2

        // Post decrement
        a = 5;
        cout << "a value Post decrement: " << a << endl; //5
        b = a--;
        cout << "b value after a-- : " << b << endl; //5
        cout << "a value after a-- : " << a << endl; //4

        // Pre decrement
        a = 5;
        cout << "a value Pre decrement: "<< a<<endl; //5
        b = --a;
        cout << "b value after --a : " << b << endl; //4
        cout << "a value after --a : " << a << endl; //4
```

```
            return 0;
 }
```

Program No. 12

Create a Base class that consists of private, protected and public data members and member functions. Try using different access modifiers for inheriting Base class to the Derived class and create a table that summarizes the above three modes (when derived in public, protected and private modes) and shows the access specifier of the members of base class in the Derived class.

```
// C++ program to demonstrate the working of public inheritance

#include <iostream>
using namespace std;

class Base {
  private:
   int pvt = 1;

  protected:
   int prot = 2;

  public:
   int pub = 3;

   // function to access private member
   int getPVT() {
       return pvt;
   }
};

class PublicDerived : public Base {
  public:
   // function to access protected member from Base
   int getProt() {
       return prot;
   }
};

int main() {
   PublicDerived object1;
```

```
    cout << "Private = " << object1.getPVT() << endl;
    cout << "Protected = " << object1.getProt() << endl;
    cout << "Public = " << object1.pub << endl;
    return 0;
}
```

Program No. 13

Create a class called Student that contains the data members like age, name, enroll_no, marks. Create another class called Faculty that contains data members like facultyName, facultyCode, salary, deptt, age, experience, gender. Create the function display() in both the classes to display the respective information. The derived Class Person demonstrates multiple inheritance. The program should be able to call both the base classes and displays their information. Remove the ambiguity (When we have exactly same variables or same methods in both the base classes, which one will be called?) by proper mechanism.

Method 1

```
#include<iostream>
using namespace std;
class Student
{
        public:
                    int age;
  string name;
           string enroll_no;
                    float marks;

        void getStudent()
        {
                cout << "Enter age of student: ";
cin >> age;
        cout << "Enter Name of student: ";
cin >> name;
                cout << "Enter Enrollment Number of student: ";
cin >> enroll_no;
                cout << "Enter Marks of student: "; cin >> marks;
        }
};

class Faculty
{
        public:
                string facultyName;
                string facultyCode;
                float salary;
```

```
                    string deptt;
                    int age;
                    float experience;
                    string gender;

          void getFaculty ()
          {
             cout << "Enter Faculty Name: ";
  cin >> facultyName;
             cout << "Enter Faculty Code: ";
  cin >> facultyCode;
             cout << "Enter Faculty Salary: ";
   cin >> salary;
             cout << "Enter Faculty Department:";
   cin >> deptt;
             cout << "Enter Faculty Age:";
   cin >> age;
             cout << "Enter Faculty Experience:";
   cin >> experience;
             cout << "Enter Faculty Gender:";
  cin >> gender;
          }
};

class Person : public Student , public Faculty

//Person is derived from class Student and class Faculty
{
          public:
          void display()
          {
cout<<"Person details: = " << "\n"<< "Student Age:" << Student::age << "\n" << "Student Name:" << Student::name << "\n"<< "Student Enrollment Number:" << Student::enroll_no << "\n" << "Student Marks:" << Student::marks << "\n" <<"Faculty Name:" << Faculty::facultyName << "\n" << "Faculty Code:" << Faculty::facultyCode << "\n" << "Faculty Salary:" << Faculty::salary << "\n" << "Faculty Department:" << Faculty::deptt << "\n" << "Faculty Age:" << Faculty::age << "\n" << "Faculty Experience:" << Faculty::experience <<"\n" << "Faculty Gender:" << Faculty::gender << "\n" ;
```

```
        }
};
int main()
{
        Person obj1; //object of derived class C
        obj1.getStudent();
        obj1.getFaculty();
        obj1.display();
        return 0;
}       //end of program
```

Method 2 (using display function):

```
#include<iostream>
using namespace std;
class Student
{
        public:
                int age;
                string name;
                string enroll_no;
                float marks;

              void display()
            {
                cout << "Enter age of student: ";
                cin >> age;
                cout << "Enter Name of student: ";
                cin >> name;
                cout << "Enter Enrollment Number of student: ";
                cin >> enroll_no;
                cout << "Enter Marks of student: ";
                cin >> marks;
            }
};
class Faculty
{
        public:
               string facultyName;
               string facultyCode;
               float salary;
               string deptt;
               int age;
               float experience;
               string gender;

               void display ()
               {
```

```
                        cout << "Enter Facuty Name: ";
                        cin >> facultyName;
                        cout << "Enter Faculty Code: ";
                        cin >> facultyCode;
                        cout << "Enter Faculty Salary: ";
                        cin >> salary;
                        cout << "Enter Faculty Department:";
                        cin >> deptt;
                        cout << "Enter Faculty Age:";
                        cin >> age;
                        cout << "Enter Faculty Experience:";
                        cin >> experience;
                        cout << "Enter Faculty Gender:";
                        cin >> gender;
                    }
};
class Person : public Student , public Faculty
//Person is derived from class Student and class Faculty
{
        public:
                    void display()
                    {
cout << "Person details: = " <<"\n"<< "Student Age:"<<Student::age<<"\n"<<"Student Name:"<<Student::name<<"\n"<<"Student Enrollment Number:" << Student::enroll_no << "\n" << "Student Marks:" << Student::marks << "\n" << "Faculty Name:" << Faculty::facultyName << "\n" << "Faculty Code:" << Faculty::facultyCode << "\n" << "Faculty Salary:" << Faculty::salary << "\n" << "Faculty Department:" << Faculty::deptt << "\n" << "Faculty Age:" << Faculty::age << "\n" << "Faculty Experience:" << Faculty::experience << "\n" << "Faculty Gender:" << Faculty::gender << "\n";
                    }
};
int main()
{
         Person obj1; //object of derived class C
         obj1.Student::display();
         obj1.Faculty::display();
         obj1.display();
         return 0;
}        //end of program
```

Program No. 14

Create a base class called shape. Use this class to store two double type values that could be used to compute the area of figures. Derive two specific classes called triangle and rectangle from base shape. Add to the base class , a member function get_data() to initialize base class data members and another member function display_area() to compute and display the area of figures. Make display_area() as a virtual function and redefine this function in the derived class to suit their requirements. Using these three classes, design a program that will accept dimensions of a triangle or a rectangle interactively and display the area.

Remember the two values given as input will be treated as lengths of two sides in the case of rectangles and as base and height in the case of triangle and used as follows:

Area of rectangle = x * y Area of triangle = ½ *x*y

Theory:

Virtual Function in C++

A virtual function is a member function which is declared within a base class and is re-defined(Overridden) by a derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- **Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.**
- **They are mainly used to achieve Runtime polymorphism**
- **Functions are declared with a virtual keyword in base class.**
- **The resolving of function call is done at Run-time.**

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

//class shape
class Shape
{
public:
        double height,base;

//constructor to assign initial values to height and base
        Shape()
        {
                height=0;
                base=0;
        }

//get_data() function to get values of height and base
        void get_data()
        {
          cout<<"\nEnter height and base to compute area :";
          cin>>height>>base;
        }

//declaration of virtual function display_area()
        virtual void display_area()
        {
        }
};
//class triangle inheriting class Shape
class Triangle : public Shape
{
        public:
        //redefining function display_area()
        void display_area()
        {
                cout<<"\nArea of Triangle ="<<(height*base)/2;
        }
};

//class Rectangle inheriting class Shape
class Rectangle : public Shape
{
public:
//redefining function display_area()
void display_area()
{
        cout<<"\nArea of Rectangle = "<<height*base;
}
```

```
};
int main()
{
Shape *s;
Triangle t;
t.get_data();
s=&t;
s->display_area();
Rectangle r;
r.get_data();
s=&r;
s->display_area();
return 0;
}
```

Program No. 15

Create a base class called **CAL_VOLUME (Abstract).** Use this class to store float type values that could be used to compute the volume of figures. Derive three specific classes called **cone, hemisphere** and **cylinder** from the base **CAL_VOLUME**. Add to the base class, a member function **getdata ( )** to initialize base class data members and another member function display **volume( )** to compute and display the volume of figures. Make display **volume ( )** as a pure virtual function and redefine this function in the derived classes to suit their requirements. Using these three classes, design a program that will accept dimensions of a cone, cylinder and hemisphere interactively and display the volumes. Remember values given as input will be and used as follows:

Volume of cone = $(1/3)\pi r^2 h$

Volume of hemisphere = $(2/3)\pi r^3$

Volume of cylinder = $\pi r^2 h$

```
#include<iostream>
#include <cmath>
#include <iomanip>

const float PI = 3.14;
using namespace std;

class Cal_Volume
{
   public:
         float r,h;
      Cal_Volume()
      {
         r=0;
         h=0;
      }
```

```
        void getdata ()
        {
          cout<<"Enter radius and height:"<<endl;
          cin>>r>>h;
        }

        virtual void display_volume() =0;
};
 class Cone :public Cal_Volume
{
    public: void display_volume()
    {
        cout<<"Volume of Cone "<<(1/(float)3)*PI*r*r*h<<endl ;
    }
};
 class Hemisphere: public Cal_Volume
{
    public: void display_volume()
    {
        cout<<"Volume of Hemisphere"<<(2/(float)3)*PI*r*r*r<<endl;
    }
};

class Cylinder: public Cal_Volume
{
    public: void display_volume ()
    {
        cout<<"Volume of Cylinder "<<PI*r*r*h<<endl;
    }
};
```

```
int main()
{
   Cal_Volume *ca;
    Cone co;
    ca= &co;
    ca->getdata();
    ca->display_volume();
  Cal_Volume *cb;
    Cylinder cy;
    cb= &cy;
    cb->getdata();
    cb->display_volume();
    Cal_Volume *cc;
    Hemisphere he;
    cc= &he;
    cc->getdata();
    cc->display_volume();
     return 0;
 }
```

Program No. 16

Implement a C++ program to demonstrate and understand Diamond problem.

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

class Person {
// Data members of person
public:
        Person(int x)
        {
                cout << "Person::Person(int ) called" << endl;
        }
};

class Faculty : public Person {
// data members of Faculty
public:
        Faculty(int x):Person(x)
        {
                cout<<"Faculty::Faculty(int ) called"<< endl;
        }
};

class Student : public Person {
// data members of Student
public:
        Student(int x):Person(x)
        {
                cout<<"Student::Student(int ) called"<< endl;
        }
};
```

```
class TA : public Faculty, public Student {
public:
        TA(int x):Student(x), Faculty(x)
        {
                cout<<"TA::TA(int ) called"<< endl;
        }
};

int main()
{
        TA ta1(30);
}
```