Q1.

Write a C++ program to create a function void arraySort(int A[], int p, int B[], int q) .Given two sorted arrays A and B of size p and q, define function arraySort() to merge elements of A with B by maintaining the sorted order i.e. fill A with first p smallest elements and fill B with remaining elements

```
#include<iostream>

using namespace std;


void sort(int temp[], int size)

{

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

    {

        for(int j = 0; j < size-i-1; j++)

        {

            if (temp[j] > temp[j+1])

            {

                int var = temp[j+1];

                temp[j+1] = temp[j];

                temp[j] = var;

            }

        }

    }

}


int main()
```

```
{
    int m, n;

    cout << "Enter size of first array : ";
    cin >> m;
    cout << "Enter size of second array : ";
    cin >>n;

    int a[m], b[n], merge[m+n];
    cout<<"Enter elements of first array : " ;
    for(int i = 0; i < m; i++)
    {
        cin >> a[i];
        merge[i] = a[i];
    }
    cout<<"Enter elements of second array : " ;
    for(int i = 0; i < n; i++)
    {
        cin >> b[i];
        merge[i+m] = b[i];
    }

    sort(merge, m+n);

    for(int i = 0; i < m+n; i++)
    {
        if (i < m)
            cout << merge[i] << " ";
        else if (i == m)
            cout << endl << merge[i] << " ";
        else
            cout << merge[i] << " ";
    }

    return 0;
}
```

OUTPUT

```
Enter size of first array : 6

Enter size of second array : 3

Enter elements of first array : 1 5 6 7 8 10

Enter elements of second array : 2 4 9

1 2 4 5 6 7

8 9 10
```

Q2.

Write a C++ program to input any string and arrange all characters in following order. Digit + Uppercase + Lowercase + Special Characters.

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


int main()
{
  string str;
  string digit = "", lower = "", upper = "", spcl = "";
  cout<<"Enter the string : " ;
  getline(cin, str);


  int i = 0;


  while (i < str.length())
  {
    if (str.at(i) >= 'A' and str.at(i) <= 'Z')
      upper += str.at(i);


    else if (str.at(i) >= 'a' and str.at(i) <= 'z')
      lower += str.at(i);


    else if (str.at(i) >= '0' and str.at(i) <= '9')
```

```
            digit += str.at(i);

        else
            spcl += str.at(i);

        i++;
    }

    cout << "Original = " << str << endl;

    cout << "After Arranging = "<< (digit + upper + lower + spcl) << endl;

    return 0;
}
```

OUTPUT

```
Enter the string : GrAphic567E#@RA

Original = GrAphic567E#@RA

After Arranging = 567GAERArphic#@
```

Q3.

Define a class employee having the following description:

DATA MEMBERS	**DESCRIPTION**
Pan	To store personal account number
Name	To store name
Taxincome	To store annual taxable income
Tax	To store tax that is calculates

MEMBER FUNCTIONS	**DESCRIPTION**
InputInfo()	Store the pan number, name, taxableincome
TaxCalc()	Calculate the tax for an employee
DisplayInfo()	Output details of an employee

TOTAL ANNUAL TAXABLE INCOME	**TAX RATE**
Up 250000	No tax
From 250000 to 300000	10% of the income exceeding 250000

```
#include <iostream>

using namespace std;



class employee

{

   int pan;

   string name;

   float taxincome;

   float tax;

public:

   void InputInfo()

   {
    cout<<"Enter Pan Number : " ;

    cin>> pan;

    cout<<"Enter Name : ";

    cin>>name;

    cout<<"Enter Taxable Income : ";

    cin>>taxincome;

}


void TaxCalc()

{

    if (taxincome  < 250000)

    {

        cout<<"No tax is to be paid : ";

        tax=0;

        taxincome+=tax;

    }

    else if ( taxincome > 250000 && taxincome <=300000)

    {

        //tax=0.1;

        taxincome = taxincome + 0.1*(taxincome - 250000);

    }

    else if ( taxincome > 300000 && taxincome <= 400000)

    {

        taxincome  = taxincome + 0.1*(taxincome - 250000);

        taxincome = 5000 + taxincome + 0.2*(taxincome - 300000);
        }
        else if ( taxincome > 400000)
        {
            taxincome  = taxincome + 0.1*(taxincome - 250000);
            taxincome = 5000 + taxincome + 0.2*(taxincome - 300000);
            taxincome = 25000 + taxincome + 0.3*(taxincome - 400000);
        }
    }

    void DisplayInfo()
    {
        cout<<"Pan Number  : "<< pan<<endl;
        cout<<"Name : " << name << endl;
        cout<<"Taxable Income : " << taxincome<< endl;
        //cout<<"Tax : " << tax << endl;
    }
};

int main()
{
    employee e1;
    e1.InputInfo();
    e1.TaxCalc();
    e1.DisplayInfo();
}
```

OUTPUT

```
Enter Pan Number : 1001

Enter Name : Amit

Enter Taxable Income : 270000

Pan Number  : 1001

Name : Amit

Taxable Income : 272000
```

Q4.

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 (ASSUME)

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

```
#include<iostream>

using namespace std;


class bankAcc

{

    string name;

    int accountNo;

    string type;

    float balance;

public:

    void assignVal(int accNo,string n,string t,float bal)

    {
```

```
    accountNo=accNo;

    name=n;

    type=t;

    balance=bal;

}


void display()

{

    cout<<"Account number of the user is : "<<accountNo<<endl;

    cout<<"Name of the user is : "<<name<<endl;

    cout<<"Type of account of the user is : "<<type<<endl;

    cout<<"Balance of the user is : "<<balance<<endl;



}


void deposit()

{

    float deposit;

    cout<<"Enter the amount to be deposited : ";

    cin>>deposit;



    balance+=deposit;

}


void withdraw()
```

```
    {
        float withdraw;
        cout<<"Enter amount to be withdrawed : ";
        cin>> withdraw;
        balance-=withdraw;
    }
};

int main()
{
    string n,t;
    int accNo;
    float bal;

    cout<<"Enter account number : ";
    cin>>accNo;
    cout<<"Enter name : ";
    getline(cin,n);
    getline(cin,n);
    cout<<"Enter type of account : ";
    getline(cin,t);
    cout<<"Enter balance : ";
    cin>>bal;

    bankAcc cust1;
    cust1.assignVal(accNo,n,t,bal);

    cust1.deposit();

    cust1.withdraw();

    cust1.display();

    return 0;


}
```

OUTPUT

```
Enter account number : 1001

Enter name : Suresh

Enter type of account : Savings

Enter balance : 50000

Enter the amount to be deposited : 5000

Enter amount to be withdrawed : 1000

Account number of the user is : 1001

Name of the user is : Suresh

Type of account of the user is : Savings

Balance of the user is : 54000
```

Q5.

Imagine a tollbooth with a class called TollBooth. The two data itemsare 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

{

   public:

   unsigned int cars;

   double amount;


   tollbooth()

   {

      cars=0;

      amount=0;

   }


   void payingCar()

   {

      cars++;
```

```
        amount+=0.5;
    }

    void nonpayingCar()
    {
        cars++;
    }

    void display()
    {
        cout<<"Total number of cars are : " << cars<<endl;
        cout<<"Total amount collected is : " <<amount<<endl;
    }
};

int main()
{
    tollbooth t;
    char key='y';
    int choice;
    //t.display();
    do
    {
        cout<<"Enter 1 to record paying Car\nEnter 2 to record non paying car\nEnter y to exit and
display\n";
```

```
        cin>>choice;
        switch(choice)
        {
        case 1:
            {
                t.payingCar();
            }break;
        case 2:
            {
                t.nonpayingCar();
            }break;
        case 3:
            {
                t.display();
                key='x';
            }
        }
        if(key=='y' || key=='Y')
        {
            cout<<"Press y to continue or exit :  " ;
        cin>>key;
        }
    }while(key == 'y' || key == 'Y');
}
```

OUTPUT

```
Enter 1 to record paying Car

Enter 2 to record non paying car

Enter 3 to exit and display

1

Press y to continue or exit :  y

Enter 1 to record paying Car

Enter 2 to record non paying car

Enter 3 to exit and display

1

Press y to continue or exit :  y

Enter 1 to record paying Car

Enter 2 to record non paying car

Enter 3 to exit and display

2

Press y to continue or exit :  y

Enter 1 to record paying Car

Enter 2 to record non paying car

Enter 3 to exit and display

3

Total number of cars are : 3

Total amount collected is : 1
```

Q6.

Create a class called Time that has separate int member data for hours, minutes and seconds. One function 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 addTime() 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()

   {

      hours=minutes=seconds=0;

   }

   time(int h, int m , int s)

   {

      hours=h;

      minutes=m;

      seconds=s;

   }
```

```
 void display()
{
   cout<<hours << " : "<< minutes << " : "<<seconds<<endl;
}

void addTime(time t1, time t2)
{
   int temp=0,temp2=0;
   seconds=t1.seconds + t2.seconds;
   if(seconds>59)
   {
      seconds=seconds-60;
      minutes++;
      temp++;
   }
   minutes=t1.minutes+ t2.minutes+temp;
   if(minutes>59)
   {
      minutes=minutes-60;
      hours++;
      temp2++;
   }
   hours=t1.hours + t2.hours+ temp2;
```

```
    }
};

int main()
{
    time t1(5,40,50),t2(7,2,60); //13: 11 : 30
    time t3;
    t1.display();
    t2.display();
    printf("Adding them \n");
    t3.addTime(t1,t2);
    t3.display();
    return 0;
}
```

OUTPUT

```
5 : 40 : 50

7 : 2 : 60

Adding them

12 : 43 : 50
```

Q7.

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

{

   private:

      float savingsBalance;

public:

   SavingsAccount(){}

   SavingsAccount(int value);

   static float annualInterestRate;

   void calculateMonthlyInterest();

   static void modifyIntererestRate(float value);

   float GetBalance() const

   {

       return 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;

}
```

OUTPUT

```
Saver 1 Savings Balance: $2666.67

Saver 2 Savings Balance: $4000


Saver 1 Savings Balance: $3777.78

Saver 2 Savings Balance: $5666.67
```

Q8.

Define a class named Test with following description:

Data Members	
Private string str	
Constructor	**Description**
Test(string)	Constructor will store value in string str
Friend Function	**Description**
Void printeven(Test ob)	This friend function will print all even position characters

Write a C++ program to print all even position characters using friend function.

```
#include<iostream>

#include<cstring>

using namespace std;


class test

{

   string str;

public :




   test(string s)

   {

      str=s;

   }


   friend void printeven(test);

};
```

```
void printeven(test t)
    {
        int len=t.str.length();
        //cout<<"Length is : " << len;
        for (int i=0;i<len ;i++)
        if(i%2==0)
            cout<<t.str[i];
    }

int main()
{
    string str;
    cout<<"Enter the string : ";
    cin>>str;

    test t(str);
    cout<<"Printing only even terms we have : ";
    printeven(t);
    return 0;
}
```

OUTPUT

```
Enter the string : graphicera

Printing only even terms we have : gahcr
```

Q9.

Write a C++ program to create a class called OverDemo and overload teststring() function. void teststring(string,int)

extract the number of characters from the right side of the passing string. void teststring(string)

Check whether passing strings is palindrome or not.

In main function invoke all member functions and perform the above task.

```
#include <iostream>

#include <bits/stdc++.h>


using namespace std;


class OverDemo

{

public:

  void teststring(string S)

   {

           transform(S.begin(), S.end(), S.begin(), ::toupper);


        string temp=S;

        reverse(S.begin(),S.end());

        if(S==temp)

            cout<<"String is palindrome\n"<<endl;

        else

          cout<<"String is not palindrome\n"<<endl;

   }
```

```
    void teststring(string S,int num)
    {
      int length = S.length();
      for(int i=length-num;i<=length;i++)
           cout<<S[i];
    }
};

int main()
{
   OverDemo obj;
   cout<<"Enter the string\n";
   string str;
   getline(cin,str);
   cout<<"Enter the value of n\n";
   int n;
   cin>>n;

   obj.teststring(str);
   obj.teststring(str,n);
   return 0;
}
```

OUTPUT

```
Enter the string

abccba

Enter the value of n

3

String is palindrome

cba
```

Q10.

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

```
#include<iostream>

using namespace std;


class complex

{

   int real,imaginary;

public:

   complex ()

   {

      real=0;

      imaginary=0;

   }

   complex(int r, int i)

   {

      real=r;

      imaginary=i;

   }



   void display()

   {
      cout<<real << " + "<<imaginary << "j";
   }

    complex operator+(complex c)
   {
      complex res;
      res.real=real+c.real;
      res.imaginary=imaginary+ c.imaginary;
      return res;
   }

   complex operator-(complex c)
   {
      complex res;
      res.real=real-c.real;
      res.imaginary=imaginary - c.imaginary;
      return res;
   }
};

int main()
{
   complex c1(3,5),c2(5,6);
   c1.display();
   cout<<endl;
   c2.display();

   cout<<endl;

   complex c3;

   cout<<"Addition using operating overloading : ";

   c3=c1 +(c2);

   c3.display();

   cout<<"\n\nSubtraction using operator overloading : ";

   c3=c1 -(c2);

   c3.display();

   cout<<endl;

   return 0;

}
```

OUTPUT

```
3 + 5j

5 + 6j

Addition using operating overloading : 8 + 11j


Subtraction using operator overloading : -2 + -1j
```

Q11.

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

```
#include<iostream>

using namespace std;


class UnaryFriend

{

   int a=10;

   int b=20;

public:

   void getvalues()

   {

      cout<<"Values of a and b "<<endl;

      cout<<a<<endl<<b<<endl;

   }

   void show()

   {

         cout<<endl<<a<<endl<<b<<endl<<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();
    -x1;
    x1.show();

    x2.getvalues();
    ++x2;
    x2.show();

    x3.getvalues();
    --x3;
    x3.show();
}
```

OUTPUT

```
Values of a and b

10

20


-10

-20


Values of a and b

10

20


11

20

Values of a and b

10

20


9

20
```

Q12.

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.

```
#include<iostream>

#include<string>

using namespace std;



class student

{

    int age,enrolNo;

public:



    float marks;

    string name;

    void getStudent()

    {

        cout<<"Enter age of student : " ;

        cin>>age ;

        cout<<"Enter name of student : " ;

        cin>>name ;

        cout<<"Enter enrollment number of student : " ;

        cin>> enrolNo;
        cout<<"Enter marks of student : " ;
        cin>> marks;
    }
    friend class person;
};

class faculty
{
    public:
    string facultyName,dept,gender;
    int facultyCode,salary,age;
    float experience;

    void getFaculty()
    {
        cout<<"Enter age of faculty : ";
        cin>> age;
        cout<<"Enter name of faculty : ";
        getline(cin,facultyName);
        getline(cin,facultyName);
        cout<<"Enter code of faculty : ";
        cin>> facultyCode;
        cout<<"Enter department of faculty : ";
        cin>>dept ;
        cout<<"Enter gender of faculty : ";
        cin>> gender;

        cout<<"Enter experience of faculty : ";

        cin>> experience;

        cout<<"Enter salary of faculty : ";

        cin>>salary ;

    }

};


class person : public student ,public faculty

{

public :

    void display()

    {

        cout<< "Student age : "<<student::age <<endl<< "Student Name: "<<student::name<<endl<< "Student enrollment number: "<<student::enrolNo<<endl<< "Student marks : "<<student::marks<<endl;

        cout<< "Faculty age : "<<faculty::age <<endl<< "Faculty Name: "<<facultyName<<endl<<"Faculty code : "<<facultyCode<<endl<<"Faculty department : "<<dept<<endl<<"Faculty gender : "<<gender<<endl<<"Faculty experience : "<<experience<<endl<<"Faculty salary : "<<salary<<endl;



    }

};


int main()

{

    person ob1;
    ob1.getStudent();

    ob1.getFaculty();

    ob1.display();

}
```

OUTPUT

```
Enter age of student: 20

Enter name of student: Amit

Enter enrollment number of student: 2002

Enter marks of student: 40

Enter age of faculty: 45

Enter name of faculty: Raj

Enter code of faculty: 6006

Enter department of faculty: CSE

Enter gender of faculty: Male

Enter experience of faculty: 15

Enter salary of faculty: 100000

Student age: 20

Student Name: Amit

Student enrollment number: 2002

Student marks: 40

Faculty age: 45

Faculty Name: Raj

Faculty code: 6006

Faculty department CSE

Faculty gender: Male

Faculty experience: 15

Faculty salary: 100000
```

Q13.

Implement a C++ program to demonstrate and understand Diamond problem. (Virtual base Class).

```
#include<iostream>

using namespace std;


class Person

{

public :

   Person(int x)

   {

      cout<<"Person::Person called "<<endl;

   }

};



class Faculty : public Person

{

public :

   Faculty(int x): Person( x)

   {

      cout<<"Faculty:Faculty called "<<endl;

   }
};

class Student: public Person
{
public :
    Student(int x): Person(x)
    {
        cout<<"Student:Student called "<<endl;
    }
};

class TA:public Student,public Faculty
{
public :
    TA(int x):Student(x),Faculty(x)
    {
        cout<<"TA:TA called "<<endl;
    }
};
int main()
{
    TA tal(30);
    return 0;
}
```

OUTPUT

```
Person::Person called

Student:Student called

Person::Person called

Faculty:Faculty called

TA:TA called
```

Q14.

Write a C++ program to implement pure virtual function with following details:

Create a base class Temperature

Data members:

Float temp;

Function members

void setTempData(float)

virtual void changetemp()

Sub Class Fahrenheit (subclass of Temperature)

Data members:

Float ctemp;

Function members

Override function changetemp() to convert Fahrenheit temperature into degree Celsius by using formula C=5/9*(F-32) and display converted temperature

Sub Class Celsius (subclass of Temperature)

Data members:

Float ftemp;

Function members

Override function changetemp() to convert degree Celsius into Fahrenheit temperature by using formula F=9/5*c+32 and display converted temperature.

```
#include<iostream>

using namespace std;


class temperature

{
```

```
public :
    float temp;
    void setTempData(float f)
    {
        temp=f;
    }

    virtual void changetemp()
    {


    }
};

class fahrenheit : public temperature
{
public :
    float ctemp;
    void changetemp()
    {
        ctemp=((temp -32)*5)/9;
        cout<<"Fahrenheit to Celsius is : "<<ctemp<<endl;
    }
};

class celsius : public temperature
{
```

```
public :
    float ftemp;
    void changetemp()
    {
        ftemp=(temp*9)/5 + 32;
        cout<<"Celsius to Fahrenheit is : "<<ftemp<<endl;
    }
};

int main()
{
    temperature *t;
    fahrenheit f;
    t=&f;
    t->setTempData(100);
    t->changetemp();

    celsius c;
    t=&c;
    t->setTempData(37.778);
    t->changetemp();
}
```

OUTPUT

```
Fahrenheit to Celsius is : 37.7778

Celsius to Fahrenheit is : 100
```

Q15.

Create a base class called CAL_AREA(Abstract). Use this class to store float type values that could be used to compute the volume of figures. Derive two specific classes called cone, hemisphere and cylinder from the base CAL_AREA. 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)πr2h

Volume of hemisphere = (2/3)πr3

Volume of cylinder = πr2h

```
#include<iostream>


using namespace std;

# define pi 3.142

class CAL_VOL

{

   public:

   float radius , height;


   CAL_VOL(){

      radius = 0 , height =0;

   }

   void get_data()

   {
};

        cout<<"enter radius and height \n";
        radius= 3;
        height =4;
    }

    // pure virtual function is alawys assigned 0
    virtual void display_volume() = 0; // mandatory should be defined in a all
                                         //derived class where it is to be used


};


class cone: public CAL_VOL
{

    public:

  void display_volume()
  {
      cout<<"volume of cone is " <<(pi *radius *radius*height)/3 <<endl<<endl;
  }


};
```

```
class cylinder: public CAL_VOL
{

   public:

  void display_volume()
  {
      cout<<"volume of cylinder is " <<(pi *radius *radius*height) <<endl<<endl;
  }


};

class hemisphere: public CAL_VOL
{

   public:

  void display_volume()
  {
      cout<<"volume of hemisphere is " <<(2*pi *radius *radius*radius)/3 <<endl<<endl;
  }
};

int main()
{
   //run time polymorphism is being demonstrated through this
   CAL_VOL *ca;        //pointer declared for cone
   cone co;                //instance of class cone
   ca =&co;                //to get access to base class
   ca->get_data();         //radius and height
   ca->display_volume();   //calculated as volume

   CAL_VOL  *cb;
   cylinder cy;
   cb =&cy;
   cb->get_data();
   cb->display_volume();

   CAL_VOL  *cc;
   hemisphere he;
   cc =&he;
   cc->get_data();
   cc->display_volume();

   return 0;
}
```

OUTPUT

```
enter radius and height

volume of cone is 37.704


enter radius and height

volume of cylinder is 113.112


enter radius and height

volume of hemisphere is 56.556
```

Q16.

Write a C++ to take input from a file and count number of alphabets, number of vowels and consonants.

```
#include<iostream>

#include<fstream>


using namespace std;


int main()

{

   string str; // to hold our files information per line

   ifstream myfile ("example.txt");


   if (myfile.is_open()) {

      while (getline(myfile, str)) {

         cout << "The string is : "<<str << endl;

      }

      myfile.close();

   }

   else cout << "Unable to open file";


   int vowels, consonants, alphabets;

   vowels =  consonants = alphabets =0;


   for(int i = 0; str[i]!='\0'; ++i)
```

```
    {
        alphabets++;
        if(str[i]=='a' || str[i]=='e' || str[i]=='i' ||
           str[i]=='o' || str[i]=='u' || str[i]=='A' ||
           str[i]=='E' || str[i]=='I' || str[i]=='O' ||
           str[i]=='U')
        {
            ++vowels;
        }
        else if((str[i]>='a'&& str[i]<='z') || (str[i]>='A'&& str[i]<='Z'))
        {
            ++consonants;
        }

    }

    cout << "Alphabets: " << alphabets << endl;
    cout << "Vowels: " << vowels << endl;
    cout << "Consonants: " << consonants << endl;

    return 0;
}
```

OUTPUT

```
The string is : GrapHiCEra

Alphabets: 10

Vowels: 4

Consonants: 6
```