**Name : Auti Prathamesh Sampat**

**Roll No. : 248020**

**Assignment C++ Module**

**Day1 - Lab1 :**

1.write program to test Hello World.

#include<iostream>

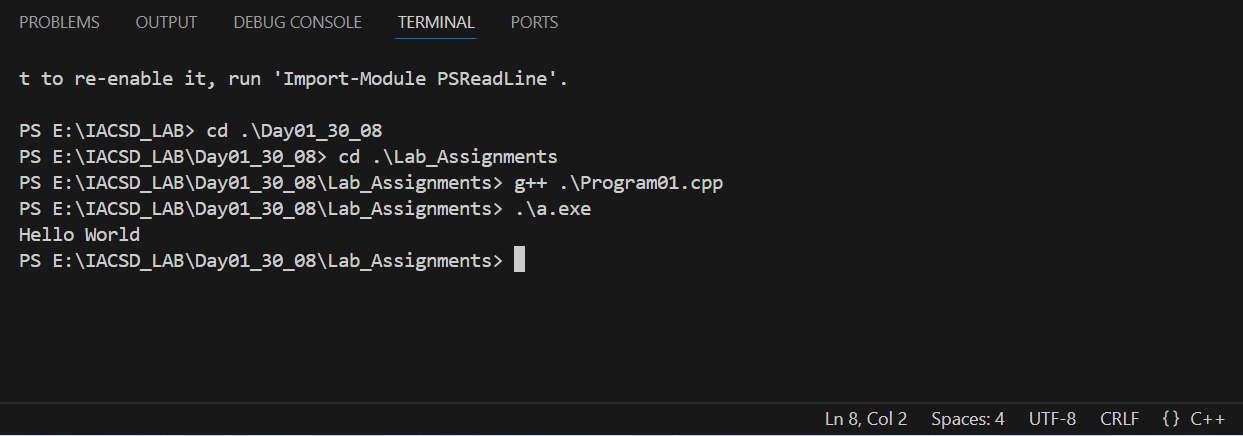
using namespace std;

int main() {

    cout<<"Hello World";

}

**Output:**

****

2:Write a program to adddition of two numbers .

#include<iostream>

using namespace std;

int main() {

    int a,b;

    cout<<"Enter 1st Number : "<<endl;

    cin>>a;

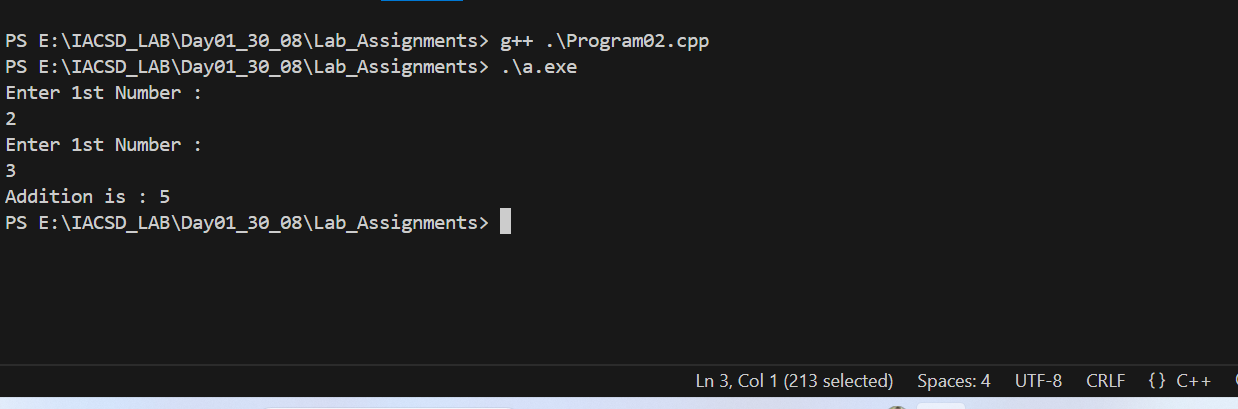
    cout<<"Enter 1st Number : "<<endl;

    cin>>b;

    cout<<"Addition is : "<<a+b;

}

Output :



3:Write a program to swap two numbers.

#include <iostream>

using namespace std;

int main()

{

    int a, b, temp;

    cout << "Enter 1st Number : " << endl;

    cin >> a;

    cout << "Enter 2nd Number : " << endl;

    cin >> b;

    cout << "Numbers before swapping: " << a << " & " << b << endl;

    temp = a;

    a = b;

    b = temp;

    cout << "Numbers after swapping: " << a << " & " << b << endl;

    int p, q;

    cout << "Enter 1st Number : " << endl;

    cin >> p;

    cout << "Enter 2nd Number : " << endl;

    cin >> q;

    cout << "Numbers before swapping: " << p << " & " << q << endl;

    p = p + q;

    q = p - q;

    p = p - q;

    cout << "Numbers after swapping: " << p << " & " << q << endl;

    int x, y;

    cout << "Enter 1st Number : " << endl;

    cin >> x;

    cout << "Enter 2nd Number : " << endl;

    cin >> y;

    cout << "Numbers before swapping: " << x << " & " << y << endl;

    x = x ^ y;

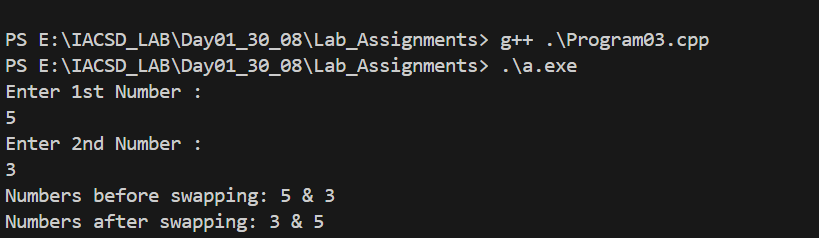
    y = x ^ y;

    x = x ^ y;

    cout << "Numbers after swapping: " << x << " & " << y << endl;

}

Output :



4. Write a program to accept an integer and check if it is even or odd.

#include <iostream>

using namespace std;

int main()

{

    int a;

    cout << "Enter Number : " << endl;

    cin >> a;

    if(a%2==0) {

        cout<<"Even Number is : "<<a;

    }

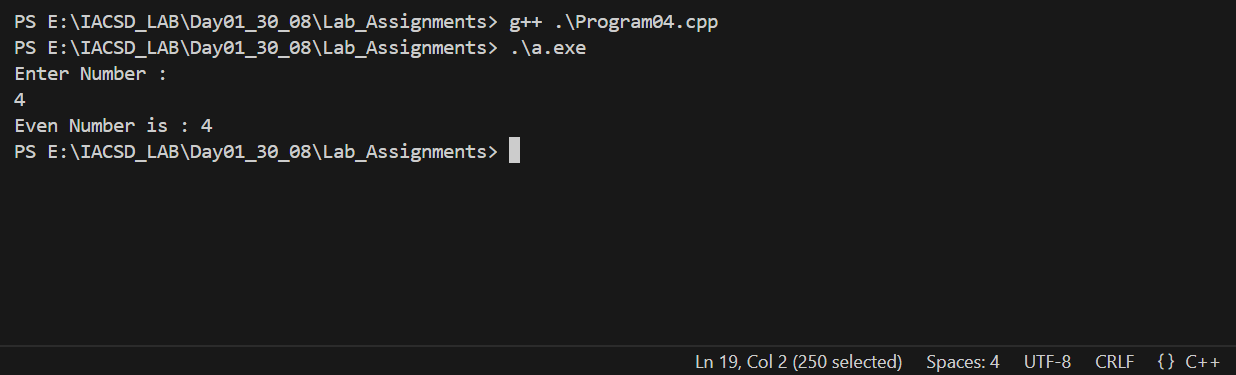
    else{

        cout<<"Not even number";

    }

}

Output :



5. Write a program to accept a number and check if it is divisible by 5 and 7.

#include<iostream>

using namespace std;

int main(){

    int num;

    cout<<"Enter the number: "<<endl;

    cin>>num;

    if(num%5==0 && num%7==0){

        cout<<"Divisible by 5 & 7....";

    }

    else{

        cout<<"Not Divisible by 5 & 7....";

    }

    return 0;

}

Output:



6. Write a program, which accepts annual basic salary of an employee and calculates and displays the

Income tax as per the following rules.

Basic: < 1, 50,000 Tax = 0

1, 50,000 to 3,00,000 Tax = 20%

> 3,00,000 Tax = 30%

#include<iostream>

using namespace std;

int main() {

    float salary,tax;

    cout<<"Enter Employee Salary : "<<endl;

    cin>>salary;

    if(salary<=150000) {

        cout<<"Tax is 0"<<"%"<<" for "<<salary<<endl;

    }

    else if(salary>150000 && salary<=300000) {

        tax = salary\*0.2;

        cout<<"Tax is 20"<<"%"<<" for "<<salary<<" so your tax is : "<<tax<<endl;

    }

    else if(salary>300000) {

         tax = salary\*0.3;

        cout<<"Tax is 30"<<"%"<<" for "<<salary<<" so your tax is : "<<tax<<endl;

    }

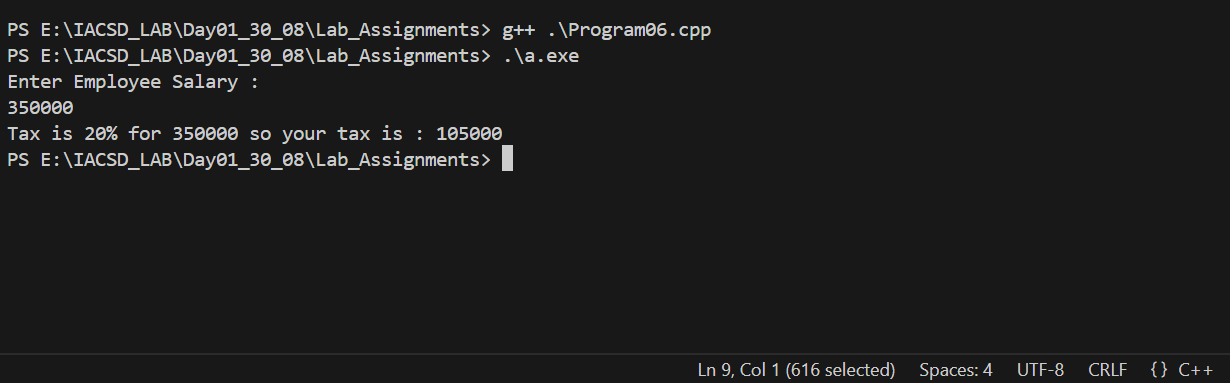
    else{

        cout<<"Enter correct amount ";

    }

}

Output:



7. (note:Daye 2 lab) Accept a lowercase character from the user and check whether the character is a vowel or consonant.

(Hint: a, e, i, o, u are vowels)

#include<iostream>

using namespace std;

int main() {

    char ch;

    cout<<"Enter lower case character : "<<endl;

    cin>>ch;

    if(ch>='a' && ch<='z'){

        if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u'){

            cout<<ch<<" is a vowel";

        }

        else{

            cout<<ch<<" is a consonant.";

        }

    }

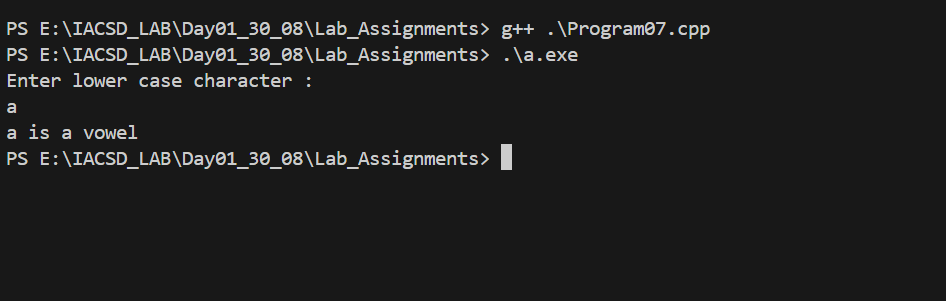
    else{

        cout<<"Enter lowercase.";

    }

}

Output:



8. Write a program to input angles of a triangle and check whether triangle is valid or not.

#include<iostream>

using namespace std;

int main() {

    int angle1,angle2,angle3,sum;

    cout<<"Enter angle1 in degree :"<<endl;

    cin>>angle1;

    cout<<"Enter angle2 in degree :"<<endl;

    cin>>angle2;

    cout<<"Enter angle3 in degree :"<<endl;

    cin>>angle3;

    sum = angle1+angle2+angle3;

    if(sum>180) {

        cout<<"Triangle is invalid"<<endl;

    }

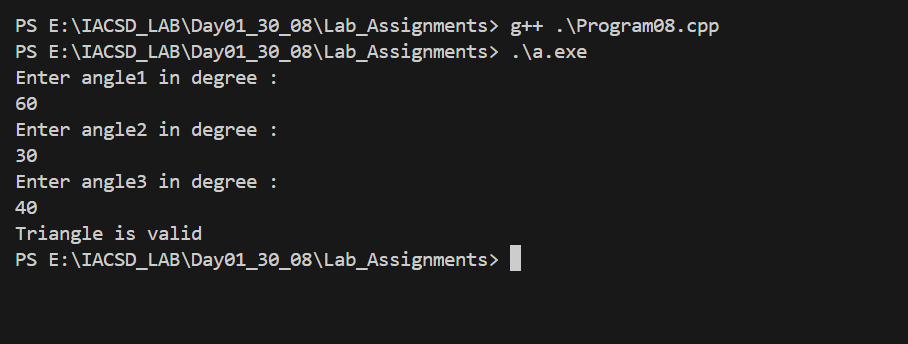
    else {

        cout<<"Triangle is valid"<<endl;

    }

}

Output:



9:Write a program to find factorial of a given number. ex:no5 fact=5\*4\*3\*2\*1=120

#include<iostream>

using namespace std;

int main() {

    int a,fact=1;

    cout<<"Enter Number : "<<endl;

    cin>>a;

    for(int i=1;i<=a;i++){

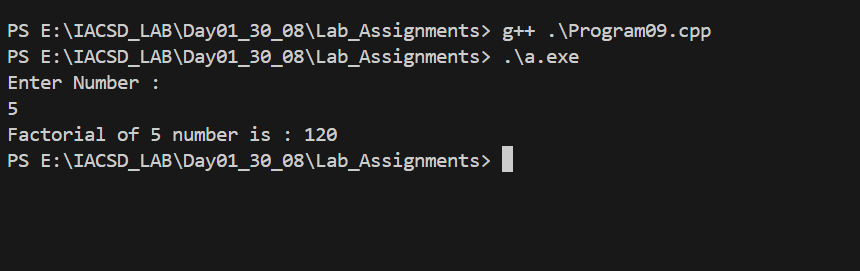
        fact\*=i;

    }

    cout<<"Factorial of "<<a<<" number is : "<<fact;

}

Output :



10:Write a program to find m to the power n. m=3 and n=4 so 3\*3\*3\*3

#include<iostream>

using namespace std;

int main() {

    int a,raised,ans=1;

    cout<<"Enter base Number : "<<endl;

    cin>>a;

    cout<<"Enter raised Number : "<<endl;

    cin>>raised;

    for(int i=1;i<=raised;i++) {

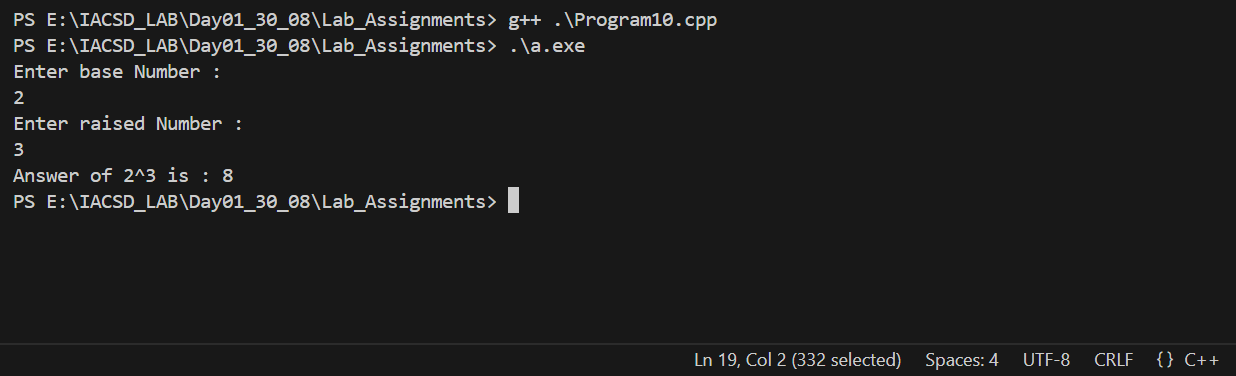
        ans=ans\*a;

    }

    cout<<"Answer of "<<a<<"^"<<raised<<" is : "<<ans;

}

Output :



11:Check if number is a prime number or not.:

#include<iostream>

using namespace std;

int main() {

    int a,count=0;

    cout<<"Enter Number : "<<endl;

    cin>>a;

    for(int i=1;i<=a;i++) {

        if(a%i==0){

            count++;

        }

    }

    if(count==2){

        cout<<"Prime Number";

    }

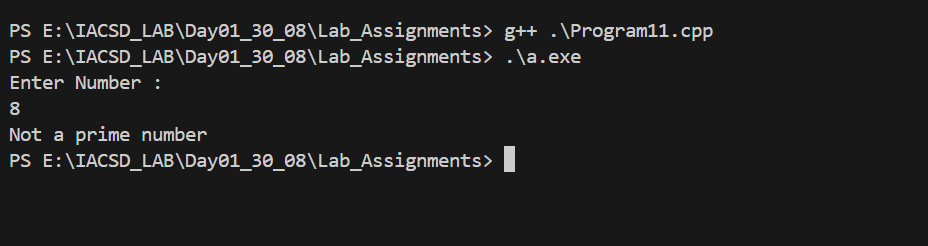
    else{

        cout<<"Not a prime number";

    }

}

Output:



12:Sum of series :

1+2+3+….+n

#include<iostream>

using namespace std;

int main() {

    int sum=0,n;

    cout<<"Enter the Number : ";

    cin>>n;

    for(int i=1;i<=n;i++) {

        sum+=i;

    }

    cout<<"Sum is : "<<sum;

    /\*

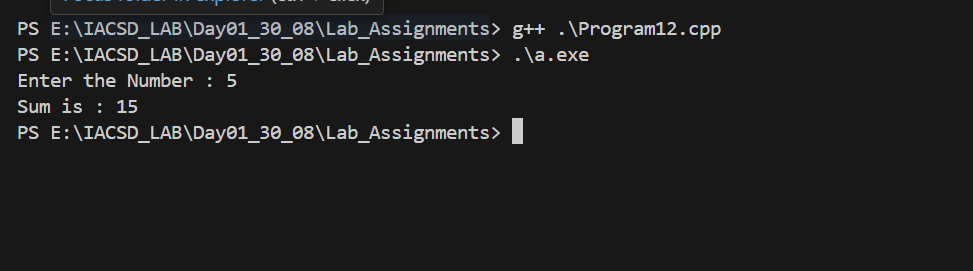
    or

    (n\*(n+1))/2

    \*/

}

Output:



13:Check whether the number is palindrome or not?

#include<iostream>

using namespace std;

int main() {

    int n,rev=0,rem;

    cout<<"Enter a number : ";

    cin>>n;

    int temp=n;

    while(temp!=0) {

        rem=temp%10;

        rev=rev\*10+rem;

        temp/=10;

    }

    if(n==rev) {

        cout<<n<<" is a palindrome";

    }

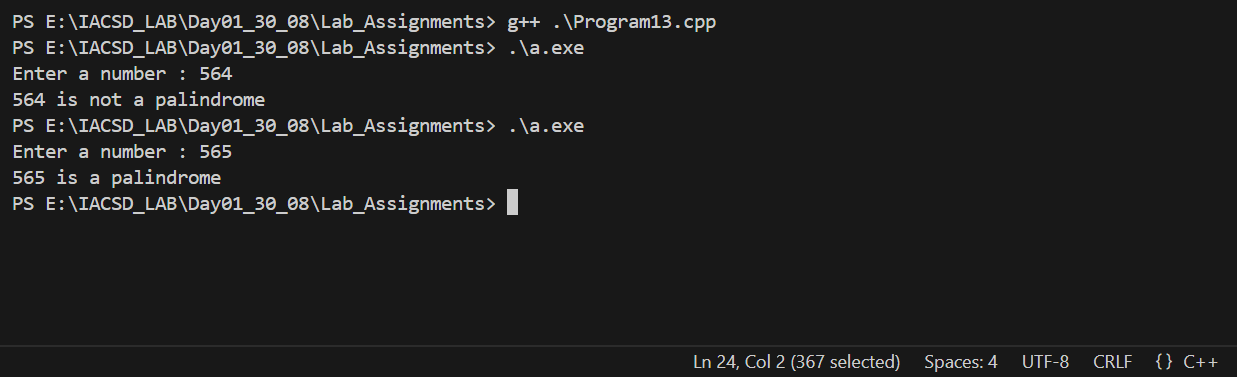
    else{

        cout<<n<<" is not a palindrome";

    }

}

Output:



14:Write a program to find sum of all even and odd numbers between 1 to n.

#include<iostream>

using namespace std;

int main() {

    int evenSum=0,oddSum=0,n;

    cout<<"Enter a number : ";

    cin>>n;

    for(int i=1;i<=n;i++) {

        if(i%2==0) {

            evenSum+=i;

        }

        else{

            oddSum+=i;

        }

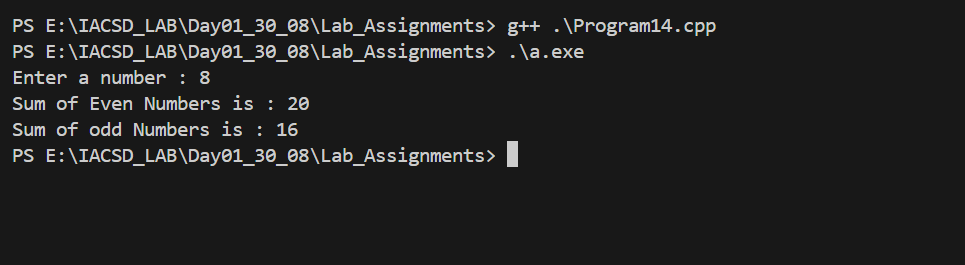
    }

    cout<<"Sum of Even Numbers is : "<<evenSum<<endl;

    cout<<"Sum of odd Numbers is : "<<oddSum;

}

Output:



15: Write a program to enter a number and print its reverse.

#include<iostream>

using namespace std;

int main() {

    int n,rev=0,rem;

    cout<<"Enter a number : ";

    cin>>n;

    int temp=n;

    while(temp!=0) {

        rem=temp%10;

        rev=rev\*10+rem;

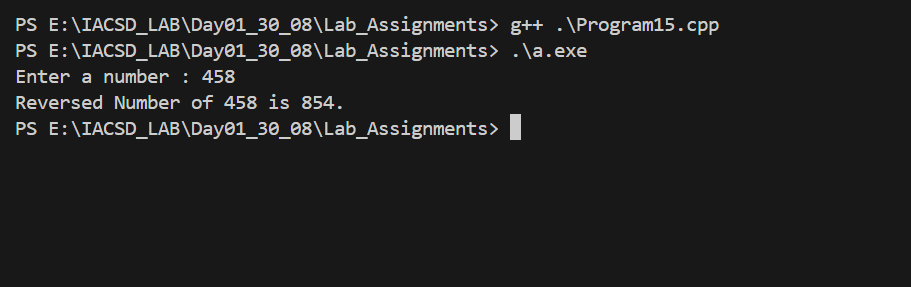
        temp/=10;

    }

    cout<<"Reversed Number of "<<n<<" is "<<rev<<". ";

}

Output:



16:Write a program to print all Prime numbers between 1 to n.

#include <iostream>

using namespace std;

int main()

{

    int n;

    cout << "Enter a number : ";

    cin >> n;

    bool flag = false;

    for (int i = 2; i <= n; i++)

    {

        for (int j = 2; j < i; j++)

        {

            if (i % j == 0)

            {

                flag = true;

                break;

            }

            else

            {

                flag = false;

            }

        }

        if (flag == false)

        {

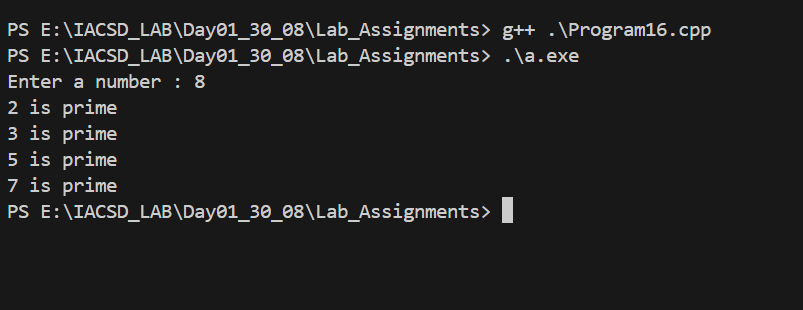
            cout << i << " is prime" << endl;

        }

    }

}

Output:



17:Write a program to check entered number is Armstrong number or not.

#include <iostream>

using namespace std;

int main()

{

    int n, count = 0, sum = 0;

    cout << "Enter a number : ";

    cin >> n;

    int temp1 = n;

    int temp2 = n;

    while (temp1 != 0)

    {

        temp1 /= 10;

        count++;

    }

    while (temp2 != 0)

    {

        int rem = temp2 % 10;

        int mult = 1;

        for (int i = 1; i <= count; i++)

        {

            mult \*= rem;

        }

        sum += mult;

        temp2 /= 10;

    }

    if (n == sum)

    {

        cout << n << " is a Armstrong Number .";

    }

    else

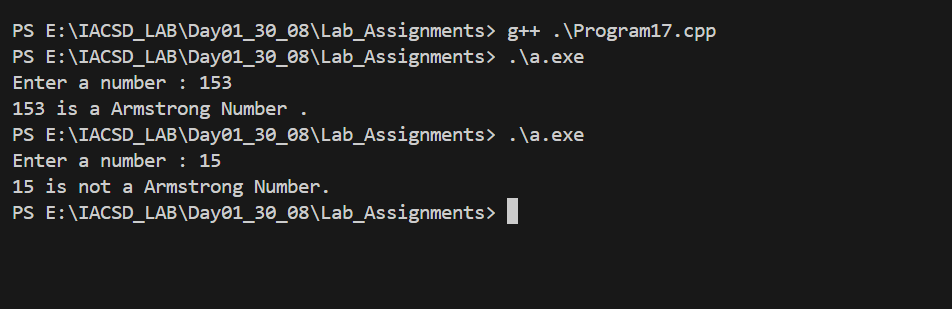
    {

        cout << n << " is not a Armstrong Number.";

    }

}

Output:



18:Write a program to find greatest of three numbers using nested if-else.

#include <iostream>

using namespace std;

int main()

{

    int num1, num2, num3;

    cout << "Enter three numbers : ";

    cin >> num1 >> num2 >> num3;

    if (num1 >= num2)

    {

        if (num1 >= num3)

        {

            cout << "Number num1 " << num1 << " is greater.";

        }

        else

        {

            cout << "Number num3 " << num3 << " is greater.";

        }

    }

    else

    {

        if (num2 >= num3)

        {

            cout << "Number num2 " << num2 << " is greater.";

        }

        else

        {

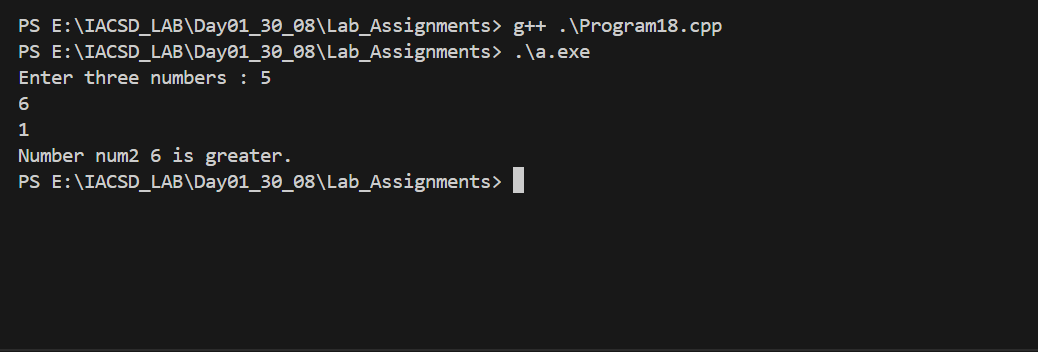
            cout << "Number num3 " << num3 << " is greater.";

        }

    }

}

Output:



19:Create menu driven program for Pizza Shop.And display total amount,

#include <iostream>

#include <string.h>

using namespace std;

int main()

{

    char name[20], pass[8], secNum[8], checkSecNum[8];

    int choice, menuChoice, qty, totalAmount = 0, amount, totalQty = 0;

    cout << "------- Welcome to the Akurdi Pizza Shop -------\n";

    cout << "Enter Your name : ";

    cin >> name;

    cout << "Enter Password : " << endl;

    cout << "----Note: Password should consist of maximum 8 characters.----" << endl;

    cin >> pass;

    cout << "Enter secure number for payment : " << endl;

    cout << "----Note: Secure number should consist of maximum 8 characters.----" << endl;

    cin >> secNum;

    // cout << "Confrm Password : ";

    // cin >> confPass;

    // if(pass==confPass){

    //     cout<<"Account created Successfully";

    // }

    cout << "Welcome, " << name << " how can we help you ?\n";

    do

    {

        cout << "Please use our following services : \n";

        cout << "1. Show Menu\n"

             << "2. Display total amount\n"

             << "3. Disply total quantity\n"

             << "4. Pay\n"

             << "5. Exit\n";

        cin >> choice;

        switch (choice)

        {

        case 1:

            do

            {

                cout << "---------Our Menu---------" << endl;

                cout << "Please choose pizza among following : " << endl;

                cout << "1. Normal Pizza - 80Rs \n"

                     << "2. Cheese Pizza - 120Rs\n"

                     << "3. Family Pizza - 300Rs\n"

                     << "4. Go to main Page.\n";

                cin >> menuChoice;

                switch (menuChoice)

                {

                case 1:

                    cout << "Enter Quantity : " << endl;

                    cin >> qty;

                    amount = qty \* 80;

                    totalAmount += amount;

                    totalQty += qty;

                    cout << "Added to cart......" << endl;

                    break;

                case 2:

                    cout << "Enter Quantity : " << endl;

                    cin >> qty;

                    totalAmount = qty \* 120;

                    totalAmount += amount;

                    totalQty += qty;

                    cout << "Added to cart......" << endl;

                    break;

                case 3:

                    cout << "Enter Quantity : " << endl;

                    cin >> qty;

                    totalAmount = qty \* 300;

                    totalAmount += amount;

                    totalQty += qty;

                    cout << "Added to cart......" << endl;

                    break;

                case 4:

                    cout << "Exiting......" << endl;

                    break;

                default:

                    cout << "Enter correct choice." << endl;

                    break;

                }

            } while (menuChoice != 4);

            break;

        case 2:

            cout << "You choose pizza of " << totalAmount << "Rs" << endl;

            break;

        case 3:

            cout << "You choose total " << totalQty << "pizzas." << endl;

            break;

        case 4:

            cout << "You have to pay total " << totalAmount << endl;

            cout << "Enter secure number for payment : ";

            cin >> checkSecNum;

            if (strcmp(checkSecNum, secNum) == 0)

            {

                cout << "Payment Successfully..." << endl;

                cout << "Now Enjoy..." << endl;

            }

            else

            {

                cout << "Enter correct secure Number ...";

            }

            break;

        case 5:

            cout << "Exiting...." << endl;

            break;

        default:

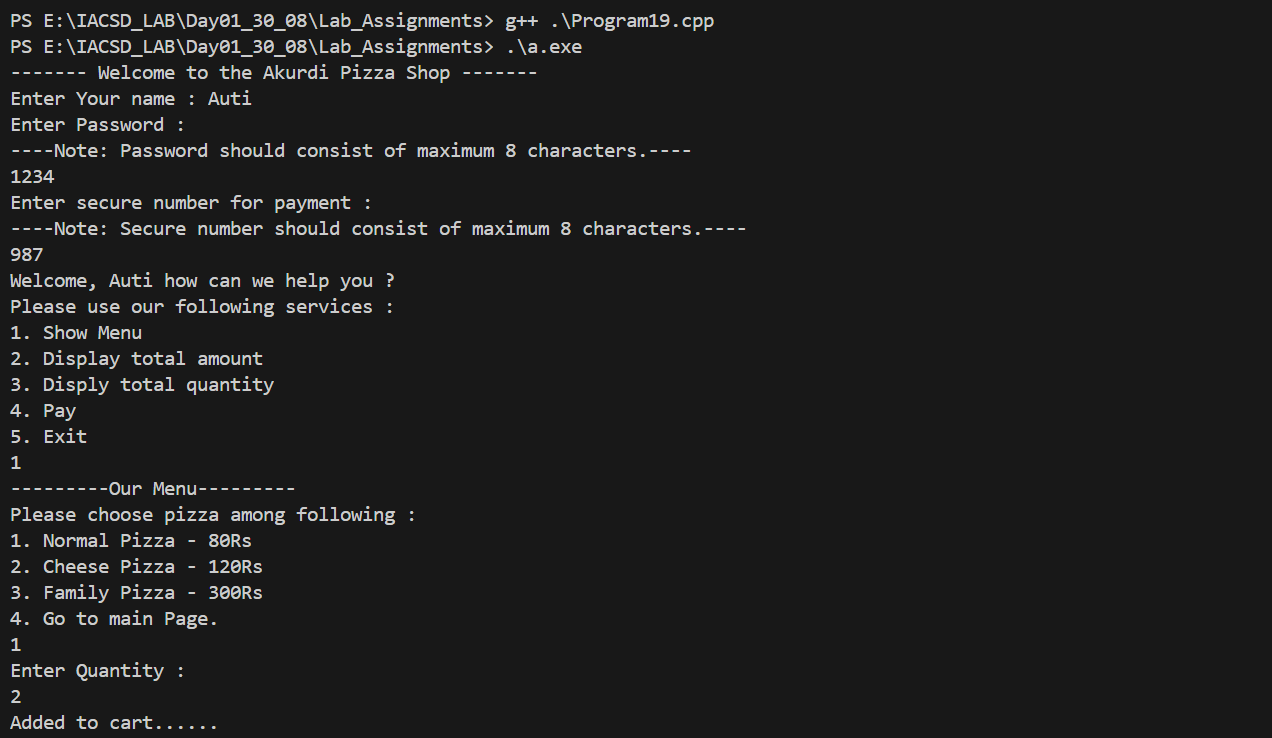
            break;

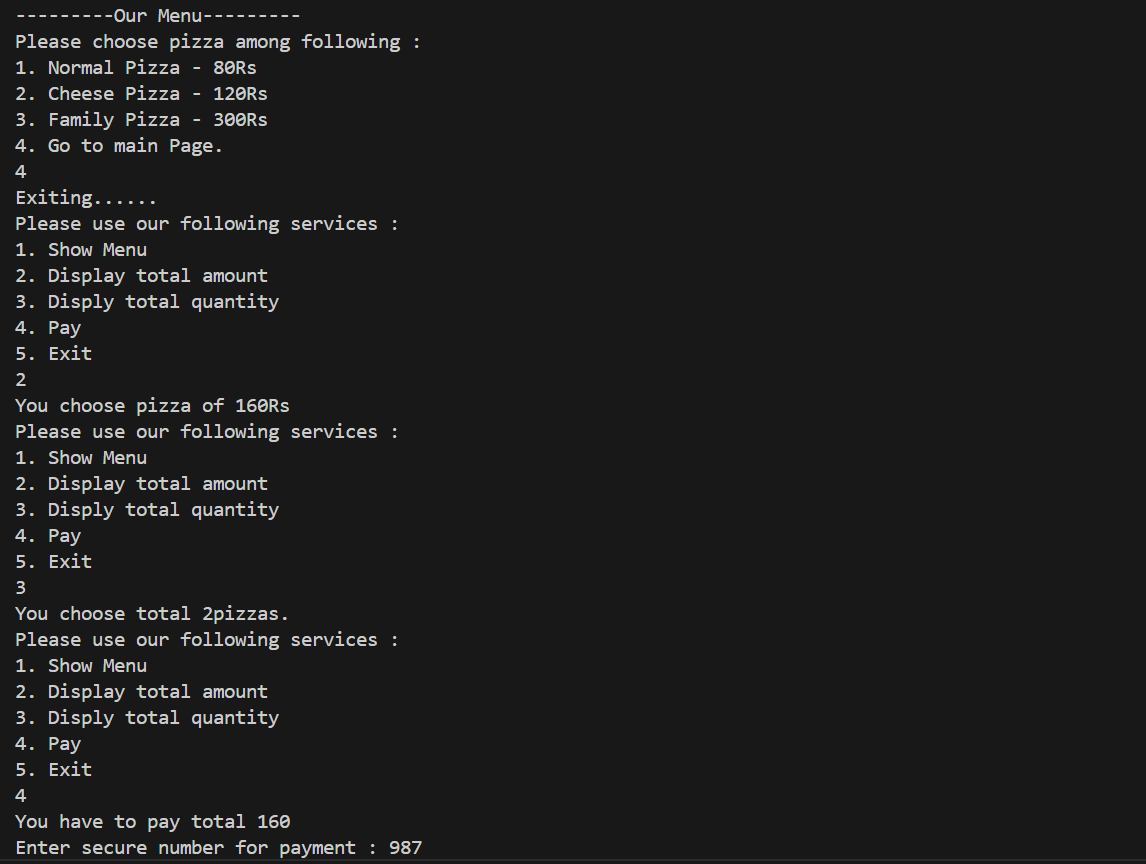
        }

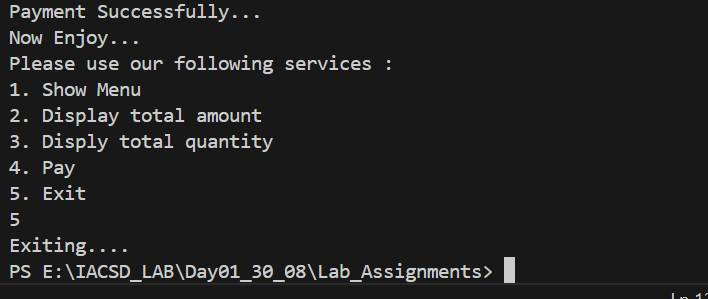
    } while (choice != 5);

}

Output:







20:Accept a single digit from the user and display it in words. For example, if digit entered is 9, display Nine.

#include <iostream>

using namespace std;

int main()

{

    int digit;

    cout << "Enter a digit : ";

    cin >> digit;

    switch (digit)

    {

    case 1:

        cout << "One";

        break;

    case 2:

        cout << "Two";

        break;

    case 3:

        cout << "Three";

        break;

    case 4:

        cout << "Four";

        break;

    case 5:

        cout << "Five";

        break;

    case 6:

        cout << "Six";

        break;

    case 7:

        cout << "Seven";

        break;

    case 8:

        cout << "Eight";

        break;

    case 9:

        cout << "Nine";

        break;

    case 0:

        cout << "Zero";

        break;

    default:

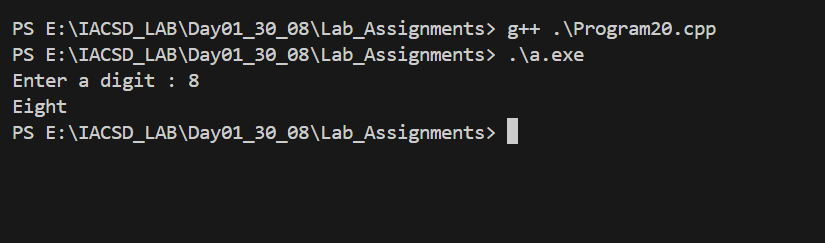
        cout << "Enter correct single digit.";

        break;

    }

}

Output:



21. Write a program, which accepts two integers and an operator as a character (+ - \* / ), performs the

corresponding operation and displays the result.

#include<iostream>

using namespace std;

int main() {

    int a,b,ans;

    char op;

    cout<<"Enter 1st number : ";

    cin>>a;

    cout<<"Enter 2nd number : ";

    cin>>b;

    cout<<"Enter operator from(+,/,\*,-) : "<<endl;

    cin>>op;

    switch (op)

    {

    case '+':

        ans = a+b;

        cout<<"Addition is : "<<ans;

        break;

    case '-':

        ans = a-b;

        cout<<"Substraction is : "<<ans;

        break;

    case '\*':

        ans= a\*b;

        cout<<"Mulitplication is : "<<ans;

        break;

    case '/':

        ans= a/b;

        cout<<"Division is : "<<ans;

        break;

    default:

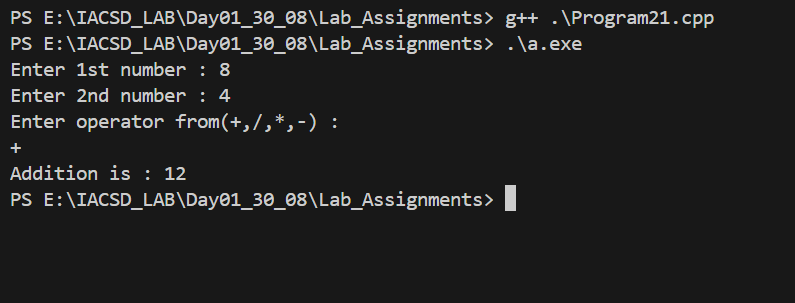
        cout<<"Enter correct operator.";

        break;

    }

}

Output:



**Day2 – Lab2 :**

**Not Given**

**Day3 – Lab3 :**

**1.**

\* Create a class Date with data members as dd, mm, yy.

\* Write a method acceptDate(inti,int j,int k) fro assigning day month year;

\* Also add the display function to print day,month,year

\* Create the object of this class in main method and invoke all the methods in that class.

#include <iostream>

using namespace std;

class Date

{

private:

    int dd;

    int mm;

    int yy;

public:

    void acceptDate(int i, int j, int k)

    {

        dd = i;

        mm = j;

        yy = k;

    }

    void displayDate()

    {

        cout << "Day is : " << dd << endl;

        cout << "Month is : " << mm << endl;

        cout << "Year is : " << yy;

    }

};

int main()

{

    Date date1;

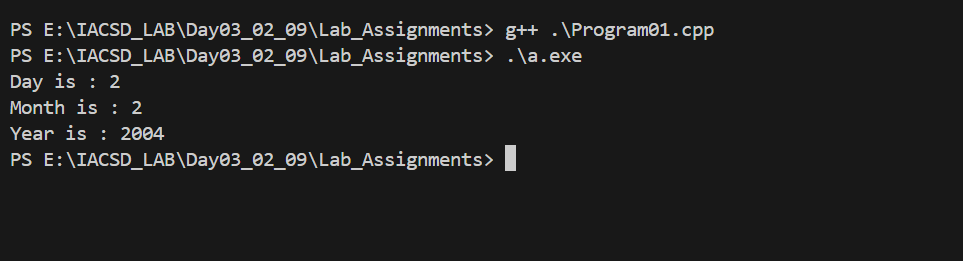
    date1.acceptDate(02, 02, 2004);

    date1.displayDate();

    return 0;

}

**Output :**



**2.**

\*Create a class Box with data members

\*create function to assign values and retrive value

\*create object of this class in main method and invoke all the methods in that class.

#include<iostream>

using namespace std;

class Box {

    private:

        string size;

        string color;

        string shape;

    public:

        void boxData(string s,string c,string sh) {

            size = s;

            color = c;

            shape = sh;

        }

        void displayBoxData() {

            cout<<"Size of Box is : "<<size<<endl;

            cout<<"Color of Box is : "<<color<<endl;

            cout<<"Shape of Box is : "<<shape;

        }

};

int main() {

    Box box1;

    string size;

    string color;

    string shape;

    cout<<"Enter size of Box e.g. big,medium,small,large : ";

    cin>>size;

    cout<<"Enter color of Box e.g. red,blue,etc. ";

    cin>>color;

    cout<<"Enter shape of Box e.g. square,rectangular,etc. ";

    cin>>shape;

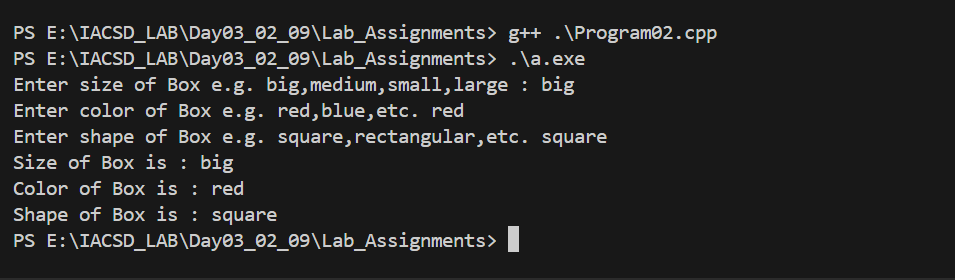
    box1.boxData(size,color,shape);

    box1.displayBoxData();

    return 0;

}

**Output :**

****

**3:**

Create a class ComplexNumber with data members real, imaginary. \*create a methods for assigning values and displaying values

write a method for retriving realNumber

ex. int getReal(){ return real;}

write same method for getImaginary

\*create an objct in main method and invoke all methods

#include<iostream>

using namespace std;

class ComplexNumber {

    private:

        int realNumber;

        int imaginaryNumber;

    public:

        void assignValues(int r,int i) {

            realNumber=r;

            imaginaryNumber=i;

        }

        void displayBoxData() {

            cout<<"Real Number is : "<<realNumber<<endl;

            cout<<"Imaginary Number is : "<<imaginaryNumber<<endl;

        }

        int getReal(){

            return realNumber;

        }

};

int main() {

    ComplexNumber cm1;

    int realNumber;

    int imaginaryNumber;

    cout<<"Enter Real Number : ";

    cin>>realNumber;

    cout<<"Enter Imaginary Number : ";

    cin>>imaginaryNumber;

    cm1.assignValues(realNumber,imaginaryNumber);

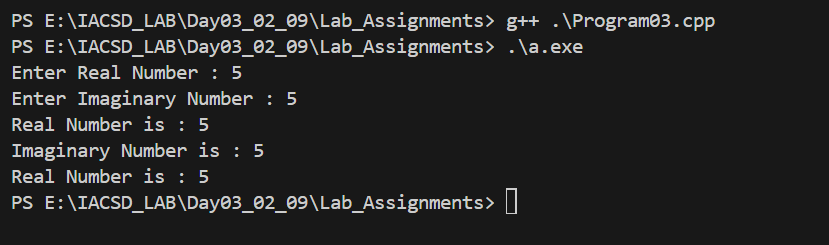
    cm1.displayBoxData();

    cout<<"Real Number is : "<<cm1.getReal();

    return 0;

}

**Output :**

****

4:Write all above classes with default and parameter constructor

and test it.

#include<iostream>

using namespace std;

class Date {

    private:

        int dd;

        int mm;

        int yy;

    public:

        Date() {

            cout<<"\n-----In Default Constructor-----"<<endl;

            dd=01;

            mm=01;

            yy=2002;

        }

        Date(int i,int j,int k) {

            cout<<"\n-----In Parameterized Constructor-----"<<endl;

            dd=i;

            mm=j;

            yy=k;

        }

        void displayDate() {

            cout<<"Day is : "<<dd<<endl;

            cout<<"Month is : "<<mm<<endl;

            cout<<"Year is : "<<yy<<endl;

        }

};

class Box {

    private:

        string size;

        string color;

        string shape;

    public:

        Box (){

            cout<<"\n-----In Default Constructor-----"<<endl;

            size = "big";

            color = "Blue";

            shape = "rectangle";

        }

        Box(string s,string c,string sh) {

            cout<<"\n-----In Parameterized Constructor-----"<<endl;

            size = s;

            color = c;

            shape = sh;

        }

        void displayBoxData() {

            cout<<"Size of Box is : "<<size<<endl;

            cout<<"Color of Box is : "<<color<<endl;

            cout<<"Shape of Box is : "<<shape<<endl;

        }

};

class ComplexNumber {

    private:

        int realNumber;

        int imaginaryNumber;

    public:

        ComplexNumber() {

            cout<<"\n-----In Default Constructor-----"<<endl;

            realNumber=2;

            imaginaryNumber=4;

        }

        ComplexNumber(int r,int i) {

            cout<<"\n-----In Parameterized Constructor-----"<<endl;

            realNumber=r;

            imaginaryNumber=i;

        }

        void displayData() {

            cout<<"Real Number is : "<<realNumber<<endl;

            cout<<"Imaginary Number is : "<<imaginaryNumber<<endl;

        }

        int getReal(){

            return realNumber;

        }

};

int main() {

    Date date;

    date.displayDate();

    Date date1(02,02,2004);

    date1.displayDate();

    Box box;

    box.displayBoxData();

    Box box1("big","red","square");

    box1.displayBoxData();

    ComplexNumber cn;

    cn.displayData();

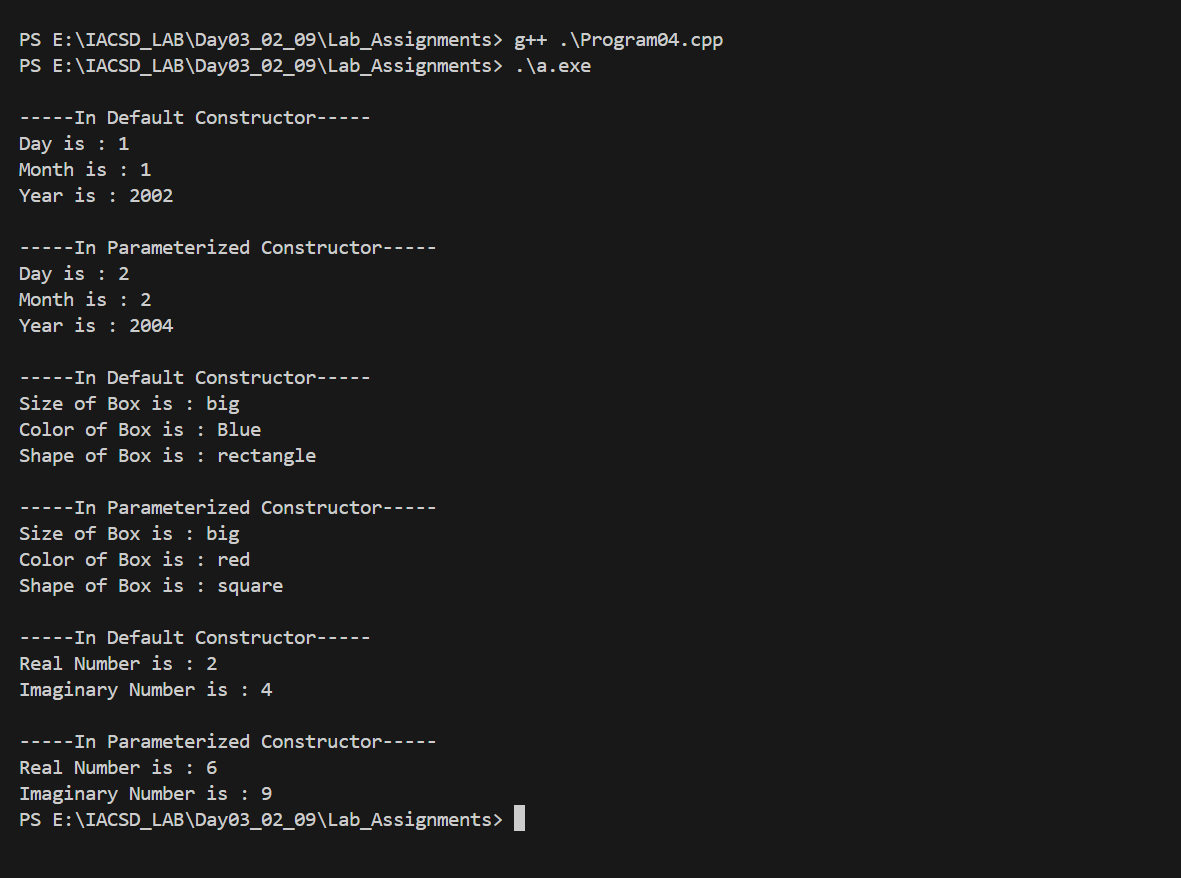
    ComplexNumber cn1(6,9);

    cn1.displayData();

    return 0;

}

**Output :**

****

**Day4 – Lab4 :**

**1.**

Create Date class with members day,month ,year.Write no argument and parameterised constructor .Create two object s and initialize them using no argument and parameterised constructor respectively.Print date using display function.

#include <iostream>

using namespace std;

class Date

{

private:

    int day, month, year;

public:

    Date()

    {

        cout << "-----Default Constructor-----" << endl;

        day = 02;

        month = 06;

        year = 2004;

    }

    Date(int day, int month, int year)

    {

        cout << "-----Parameterized Constructor-----" << endl;

        this->day = day;

        this->month = month;

        this->year = year;

    }

    void displayData()

    {

        cout << "Day is : " << day << endl;

        cout << "Month is : " << month << endl;

        cout << "Year is : " << year << endl;

    }

};

int main()

{

    Date date1;

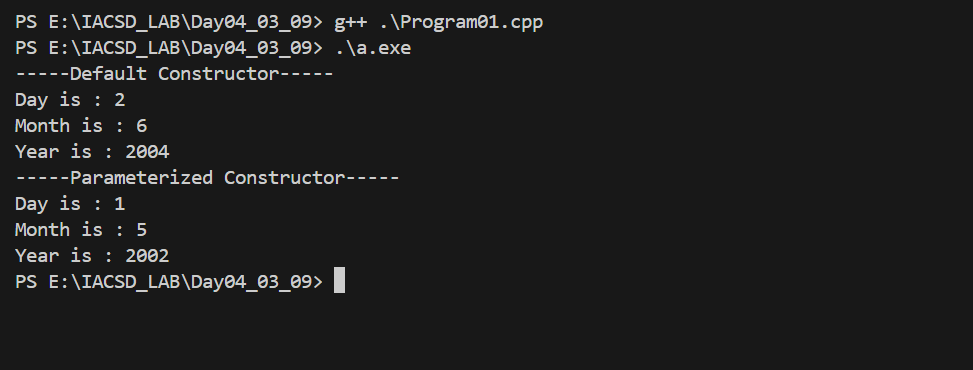
    date1.displayData();

    Date date2(01, 05, 2002);

    date2.displayData();

}

**Output :**

****

**2.**

Create Employee class with members id(int),name(string),dob(Date).Use above created Date class. Write default and parameterised constructor in Employee Class.Write accept() function to accept information and display() to display emp information.

#include <iostream>

using namespace std;

class Date

{

private:

    int day, month, year;

public:

    Date()

    {

        cout << "-----Default Constructor Date-----" << endl;

        day = 02;

        month = 06;

        year = 2004;

    }

    Date(int day, int month, int year)

    {

        cout << "-----Parameterized Constructor Date-----" << endl;

        this->day = day;

        this->month = month;

        this->year = year;

    }

    void displayData()

    {

        cout<<day<<"-"<<month<<"-"<<year<<endl;

    }

};

class Employee

{

private:

    int id;

    string name;

    Date dob;

public:

    Employee()

    {

        cout << "-----Default Constructor Employee-----" << endl;

        id = 02;

        name = "Auti";

        dob = Date();

    }

    Employee(int id, string name, Date dob)

    {

        cout << "-----Parameterized Constructor Employee-----" << endl;

        this->id = id;

        this->name = name;

        this->dob = dob;

    }

    void accept()

    {

        cout<<"Enter id : ";

        cin>>id;

        cout<<"Enter Name : ";

        cin>>name;

        int day,month,year;

        cout<<"Enter day : ";

        cin>>day;

        cout<<"Enter Month : ";

        cin>>month;

        cout<<"Enter year : ";

        cin>>year;

        dob=Date(day,month,year);

    }

    void displayData()

    {

        cout << "Id is : " << id << endl;

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

        cout << "<---- Date of Birth is ----> " << endl;

        dob.displayData();

    }

};

int main()

{

    Employee emp1;

    emp1.displayData();

    emp1.accept();

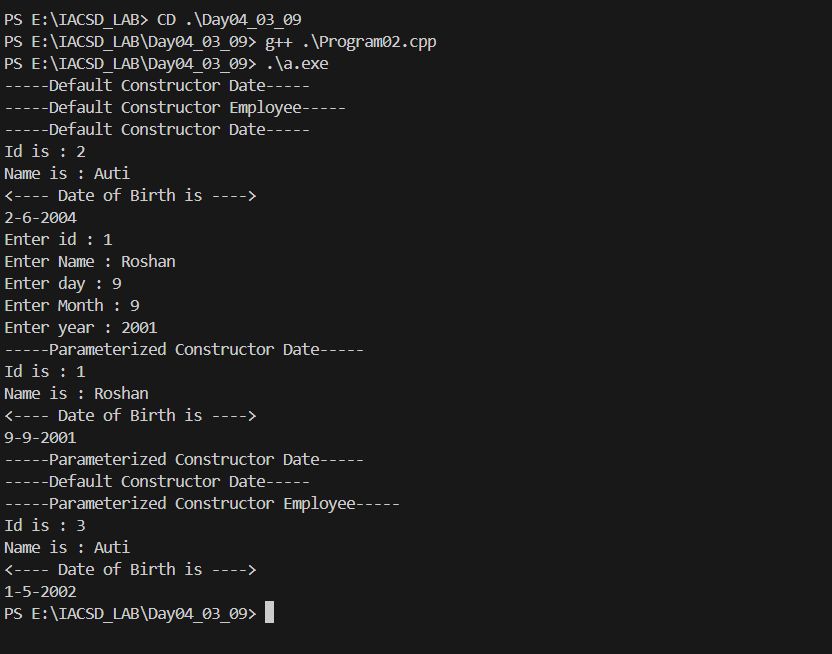
    emp1.displayData();

    Employee emp2(3,"Auti",Date(01,05,2002));

    emp2.displayData();

}

**Output :**

****

**3.**

Consider that payroll software needs to be developed for computerization of operations of an ABC organization. The organization has employees.

3.1. Construct a class Employee with following members using private access specifies:

Employee Id integer

Employee Name string

Basic Salary double

HRA double

Medical double=1000

PF double

PT double

Net Salary double

Gross Salary double

Please use following expressions for calculations://Note:Don't accept HRA,PF PT from user

\* HRA = 50% of Basic Salary

\* PF = 12% of Basic Salary

\* PT = Rs. 200

3.2. Write methods to display the details of an employee and calculate the gross and net salary.

\* Goss Salary = Basic Salary + HRA + Medical

\* Net Salary = Gross Salary – (PT + PF)

Create Object of employee class and assign values and display Details.

#include <iostream>

using namespace std;

class Employee

{

private:

    int empId;

    string name;

    double salary;

    double hra;

    double medical = 1000;

    double pf;

    double pt = 200;

    double netSalary;

    double grossSalary;

public:

    Employee()

    {

    }

    Employee(int empId, string name, double salary)

    {

        this->empId = empId;

        this->name = name;

        this->salary = salary;

    }

    double calGrossSalary()

    {

        grossSalary = salary + hra + medical;

        return grossSalary;

    }

    double calNetSalary()

    {

        netSalary = grossSalary - (pt + pf);

        return netSalary;

    }

    void displayDetails()

    {

        hra = 0.5 \* salary;

        pf = 0.12 \* salary;

        cout << "Employee Id : " << empId << endl;

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

        cout << "Employee hra is : " << hra << endl;

        cout << "Employee pf is : " << pf << endl;

        cout << "Employee Basic Salary : " << salary << endl;

        cout << "Employee Gross Salary : " << calGrossSalary() << endl;

        cout << "Employee net Salary : " << calNetSalary() << endl;

    }

};

int main()

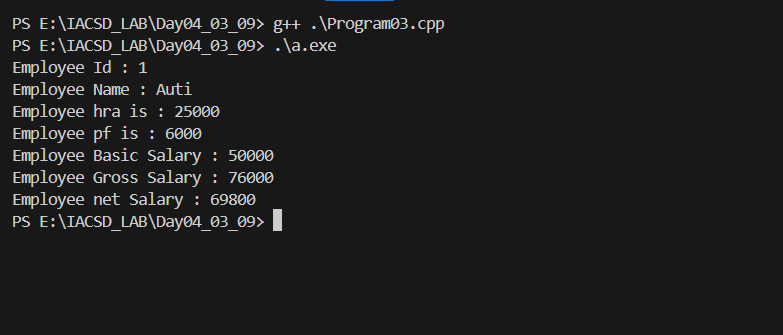
{

    Employee emp1(1, "Auti", 50000);

    emp1.displayDetails();

}

**Output :**

****

**Day 5 – Lab 5:**

1 Solve this.

Fresh business scenario to apply inheritance , polymorphism to emp based organization scenario.

Create Emp based organization structure --- Emp , Mgr , Worker

1.1 Emp state--- id(int), name, deptId , basicSalary(double)

Accept all of above in constructor arguments.

Methods ---

1.2. compute net salary ---ret 0

(eg : public double computeNetSalary(){return 0;})

1.2 Mgr state ---id,name,basic,deptId , perfBonus

Add suitable constructor

Methods ----

1. compute net salary (formula: basic+perfBonus) -- override computeNetSalary

1.3 Worker state --id,name,basic,deptId,hoursWorked,hourlyRate

Methods :

1. compute net salary (formula: = basic+(hoursWorked\*hourlyRate) --override computeNetSalary

2. get hrlyRate of the worker -- add a new method to return hourly rate of a worker.(getter)

Create suitable array to store organization details.

Provide following options

1. Hire Manager

I/P : all manager details

2. Hire Worker

I/P : all worker details

3. Display information of all employees net salary (by invoking computeNetSal),

4. Exit

----------------------------------------------------

#include <iostream>

#include <string>

using namespace std;

// Base class Employee

class Employee {

protected:

    int id;

    string name;

    int deptID;

    double basicSalary;

public:

    Employee(int id, string name, int deptID, double basicSalary)

        : id(id), name(name), deptID(deptID), basicSalary(basicSalary) {}

    // Virtual function to compute net salary (to be overridden in derived classes)

    virtual double computeNetSalary() {

        return 0;  // Base class doesn't have a formula for salary

    }

    // Virtual function to display employee details (to be overridden)

    virtual void display() {

        cout << "ID: " << id << ", Name: " << name

             << ", DeptID: " << deptID << ", Basic Salary: " << basicSalary;

    }

    // Virtual destructor for proper cleanup of derived classes

    virtual ~Employee() {}

};

// Derived class Manager

class Manager : public Employee {

private:

    double perfBonus;

public:

    Manager(int id, string name, int deptID, double basicSalary, double perfBonus)

        : Employee(id, name, deptID, basicSalary), perfBonus(perfBonus) {}

    // Override computeNetSalary for Manager

    double computeNetSalary() override {

        return basicSalary + perfBonus;

    }

    // Override display

    void display() override {

        Employee::display();

        cout << ", Performance Bonus: " << perfBonus

             << ", Net Salary: " << computeNetSalary() << endl;

    }

};

// Derived class Worker

class Worker : public Employee {

private:

    double hoursWorked;

    double hourlyRate;

public:

    Worker(int id, string name, int deptID, double basicSalary, double hoursWorked, double hourlyRate)

        : Employee(id, name, deptID, basicSalary), hoursWorked(hoursWorked), hourlyRate(hourlyRate) {}

    // Override computeNetSalary for Worker

    double computeNetSalary() override {

        return basicSalary + (hoursWorked \* hourlyRate);

    }

    // Additional method to get the hourly rate

    double getHourlyRate() {

        return hourlyRate;

    }

    // Override display

    void display()  {

        Employee::display();

        cout << ", Hours Worked: " << hoursWorked << ", Hourly Rate: " << hourlyRate

             << ", Net Salary: " << computeNetSalary() << endl;

    }

};

int main() {

    Employee\* employee[40];

    int index = 0;

    int choice;

    do {

        cout << "1. Hire Manager\n2. Hire Worker\n3. Display all employees' net salary\n4. Exit\n";

        cout << "Enter choice: ";

        cin >> choice;

        switch (choice) {

        case 1: {

            // Hire Manager

            int id, deptID;

            string name;

            double basicSalary, perfBonus;

            cout << "Enter ID, Name, DeptID, Basic Salary, Performance Bonus: ";

            cin >> id >> name >> deptID >> basicSalary >> perfBonus;

            employee[index] = new Manager(id, name, deptID, basicSalary, perfBonus);

            index++;

            break;

        }

        case 2: {

            // Hire Worker

            int id, deptID;

            string name;

            double basicSalary, hoursWorked, hourlyRate;

            cout << "Enter ID, Name, DeptID, Basic Salary, Hours Worked, Hourly Rate: ";

            cin >> id >> name >> deptID >> basicSalary >> hoursWorked >> hourlyRate;

            employee[index] = new Worker(id, name, deptID, basicSalary, hoursWorked, hourlyRate);

            index++;

            cout << "Worker Hired\n";

            break;

        }

        case 3:

            // Display all employees' net salary

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

                employee[i]->display();

            }

            break;

        case 4:

            cout << "Exiting...\n";

            break;

        default:

            cout << "Invalid choice! Try again.\n";

        }

    } while (choice != 4);

    // Clean up dynamically allocated memory

    for (int i = 0; i < index; i++) {

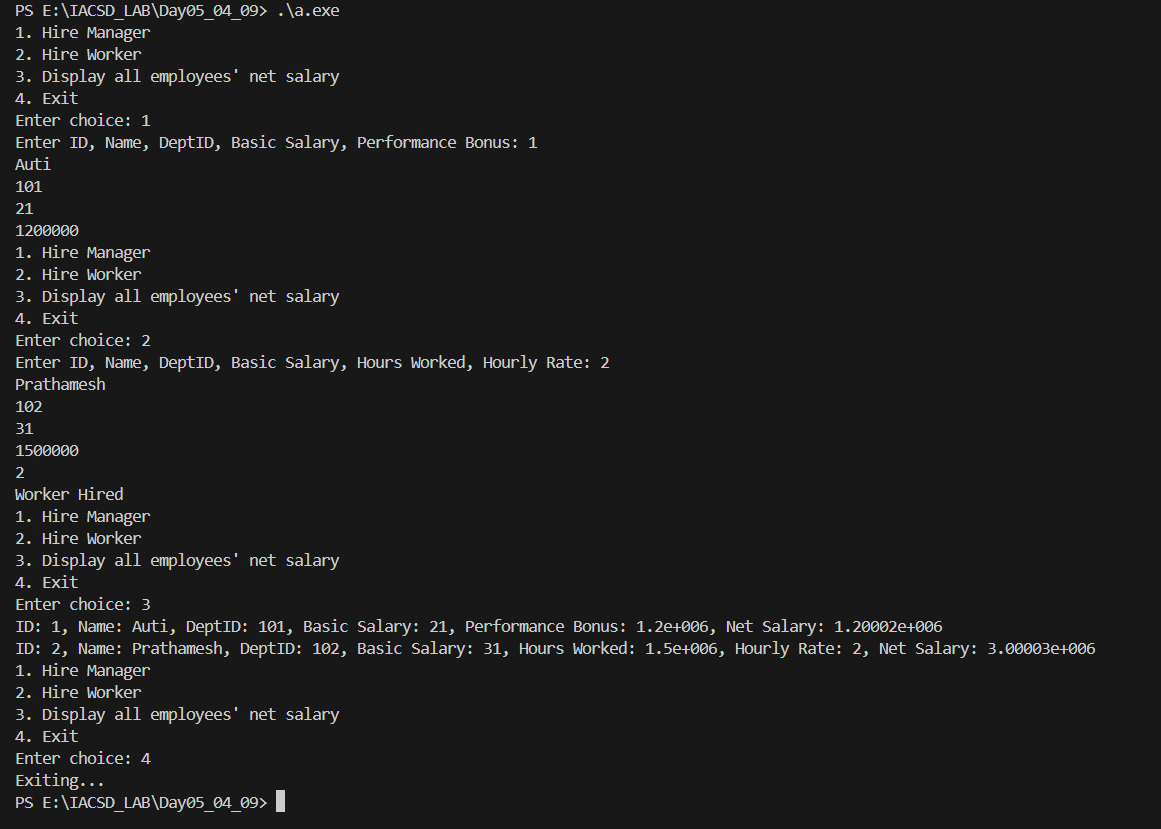
        delete employee[i];

    }

    return 0;

}

**Output :**

****

**2.**

2:Create cpp application for bank account handling.

2.1. Create a class BankAccount -- acct no(int),customer name(string),balance(double)

Add constr. (2 constrs : first to accept all details )

2.2 Add Business logic methods

Methods

public void withdraw(double amt)

public void deposit(double amt)

2.3: Create object of account class and test withdraw and deposit methods.

#include<iostream>

using namespace std;

class BankAccount{

    private:

        int accno;

        string cust\_name;

        double balance;

    public:

        BankAccount(){

            cout<<"Default....."<<endl;

        }

        BankAccount(int accno, string cust\_name,double balance){

            this->accno=accno;

            this->cust\_name=cust\_name;

            this->balance=balance;

        }

        void withdraw(double amt){

            if(amt<balance){

                balance-=amt;

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

            }

            else{

                cout<<"Insufficent balance...."<<endl;

            }

        }

        void deposite(double amt){

            if(amt>0){

                balance+=amt;

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

            }

            else{

                cout<<"Enter the sufficent balance....."<<endl;

            }

        }

};

int main(){

    BankAccount bobj(101,"Kiran",5000);

    bobj.deposite(4000);

    bobj.withdraw(290);

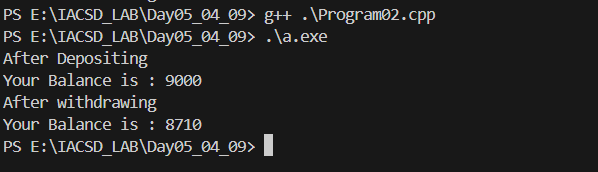
    bobj.withdraw(175);

    bobj.deposite(230);

    return 0;

}

**Output :**

****

**Day 6 – No lab**

**Day 7 – Lab 6**

1 Solve this.

Fresh business scenario to apply inheritance , polymorphism to emp based organization scenario.

Create Emp based organization structure --- Emp , Mgr , Worker

1.1 Emp state--- id(int), name, deptId , basicSalary(double)

Accept all of above in constructor arguments.

Methods ---

1.2. compute net salary ---ret 0

(eg : public double computeNetSalary(){return 0;})

1.2 Mgr state ---id,name,basic,deptId , perfBonus

Add suitable constructor

Methods ----

1. compute net salary (formula: basic+perfBonus) -- override computeNetSalary

1.3 Worker state --id,name,basic,deptId,hoursWorked,hourlyRate

Methods :

1. compute net salary (formula: = basic+(hoursWorked\*hourlyRate) --override computeNetSalary

2. get hrlyRate of the worker -- add a new method to return hourly rate of a worker.(getter)

Create suitable array to store organization details.

Provide following options

1. Hire Manager

I/P : all manager details

2. Hire Worker

I/P : all worker details

3. Display information of all employees net salary (by invoking computeNetSal),

4. Exit

----------------------------------------------------

#include <iostream>

#include <string>

using namespace std;

// Base class Employee

class Employee {

protected:

    int id;

    string name;

    int deptID;

    double basicSalary;

public:

    Employee(int id, string name, int deptID, double basicSalary)

        : id(id), name(name), deptID(deptID), basicSalary(basicSalary) {}

    // Virtual function to compute net salary (to be overridden in derived classes)

    virtual double computeNetSalary() {

        return 0;  // Base class doesn't have a formula for salary

    }

    // Virtual function to display employee details (to be overridden)

    virtual void display() {

        cout << "ID: " << id << ", Name: " << name

             << ", DeptID: " << deptID << ", Basic Salary: " << basicSalary;

    }

    // Virtual destructor for proper cleanup of derived classes

    virtual ~Employee() {}

};

// Derived class Manager

class Manager : public Employee {

private:

    double perfBonus;

public:

    Manager(int id, string name, int deptID, double basicSalary, double perfBonus)

        : Employee(id, name, deptID, basicSalary), perfBonus(perfBonus) {}

    // Override computeNetSalary for Manager

    double computeNetSalary() override {

        return basicSalary + perfBonus;

    }

    // Override display

    void display() override {

        Employee::display();

        cout << ", Performance Bonus: " << perfBonus

             << ", Net Salary: " << computeNetSalary() << endl;

    }

};

// Derived class Worker

class Worker : public Employee {

private:

    double hoursWorked;

    double hourlyRate;

public:

    Worker(int id, string name, int deptID, double basicSalary, double hoursWorked, double hourlyRate)

        : Employee(id, name, deptID, basicSalary), hoursWorked(hoursWorked), hourlyRate(hourlyRate) {}

    // Override computeNetSalary for Worker

    double computeNetSalary() override {

        return basicSalary + (hoursWorked \* hourlyRate);

    }

    // Additional method to get the hourly rate

    double getHourlyRate() {

        return hourlyRate;

    }

    // Override display

    void display()  {

        Employee::display();

        cout << ", Hours Worked: " << hoursWorked << ", Hourly Rate: " << hourlyRate

             << ", Net Salary: " << computeNetSalary() << endl;

    }

};

int main() {

    Employee\* employee[40];

    int index = 0;

    int choice;

    do {

        cout << "1. Hire Manager\n2. Hire Worker\n3. Display all employees' net salary\n4. Exit\n";

        cout << "Enter choice: ";

        cin >> choice;

        switch (choice) {

        case 1: {

            // Hire Manager

            int id, deptID;

            string name;

            double basicSalary, perfBonus;

            cout << "Enter ID, Name, DeptID, Basic Salary, Performance Bonus: ";

            cin >> id >> name >> deptID >> basicSalary >> perfBonus;

            employee[index] = new Manager(id, name, deptID, basicSalary, perfBonus);

            index++;

            break;

        }

        case 2: {

            // Hire Worker

            int id, deptID;

            string name;

            double basicSalary, hoursWorked, hourlyRate;

            cout << "Enter ID, Name, DeptID, Basic Salary, Hours Worked, Hourly Rate: ";

            cin >> id >> name >> deptID >> basicSalary >> hoursWorked >> hourlyRate;

            employee[index] = new Worker(id, name, deptID, basicSalary, hoursWorked, hourlyRate);

            index++;

            cout << "Worker Hired\n";

            break;

        }

        case 3:

            // Display all employees' net salary

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

                employee[i]->display();

            }

            break;

        case 4:

            cout << "Exiting...\n";

            break;

        default:

            cout << "Invalid choice! Try again.\n";

        }

    } while (choice != 4);

    // Clean up dynamically allocated memory

    for (int i = 0; i < index; i++) {

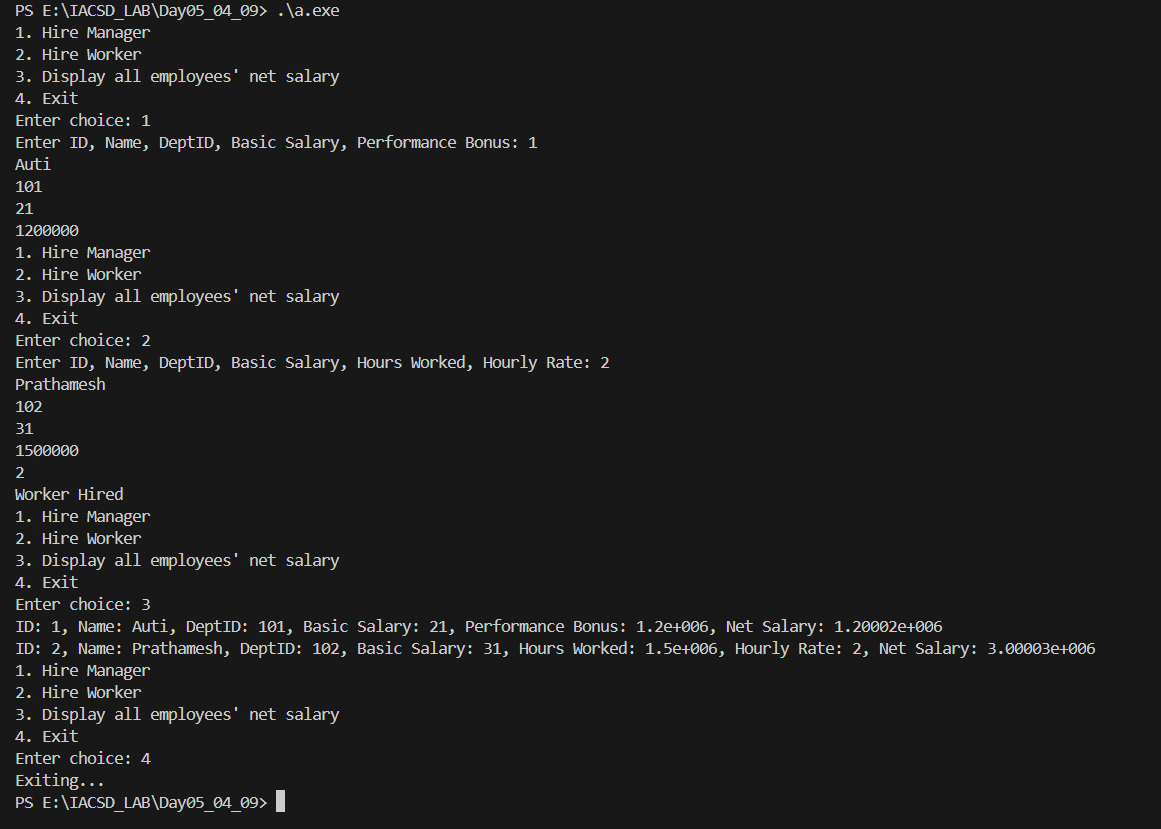
        delete employee[i];

    }

    return 0;

}

**Output :**

****

**2.**

2:Create cpp application for bank account handling.

2.1. Create a class BankAccount -- acct no(int),customer name(string),balance(double)

Add constr. (2 constrs : first to accept all details )

2.2 Add Business logic methods

Methods

public void withdraw(double amt)

public void deposit(double amt)

2.3: Create object of account class and test withdraw and deposit methods.

#include<iostream>

using namespace std;

class BankAccount{

    private:

        int accno;

        string cust\_name;

        double balance;

    public:

        BankAccount(){

            cout<<"Default....."<<endl;

        }

        BankAccount(int accno, string cust\_name,double balance){

            this->accno=accno;

            this->cust\_name=cust\_name;

            this->balance=balance;

        }

        void withdraw(double amt){

            if(amt<balance){

                balance-=amt;

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

            }

            else{

                cout<<"Insufficent balance...."<<endl;

            }

        }

        void deposite(double amt){

            if(amt>0){

                balance+=amt;

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

            }

            else{

                cout<<"Enter the sufficent balance....."<<endl;

            }

        }

};

int main(){

    BankAccount bobj(101,"Kiran",5000);

    bobj.deposite(4000);

    bobj.withdraw(290);

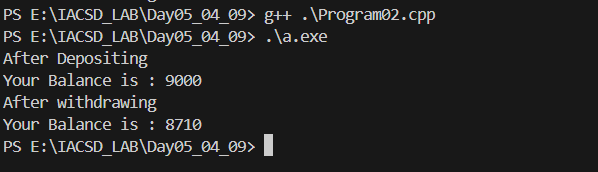
    bobj.withdraw(175);

    bobj.deposite(230);

    return 0;

}

**Output :**

****

**3.**

#include <iostream>

using namespace std;

class Shape

{

public:

    virtual void area() = 0; // Pure virtual method

    virtual ~Shape() {}      // Virtual destructor to ensure proper cleanup

};

class Rectangle : public Shape

{

    float length, breadth;

public:

    Rectangle(float length, float breadth) : length(length), breadth(breadth) {}

    void area() override

    {

        cout << "Area of Rectangle: " << length \* breadth << endl;

    }

};

class Circle : public Shape

{

    float radius;

    const float pi = 3.142;

public:

    Circle(float radius) : radius(radius) {}

    void area() override

    {

        cout << "Area of Circle: " << pi \* radius \* radius << endl;

    }

};

class Square : public Shape

{

    float side;

public:

    Square(float side) : side(side) {}

    void area() override

    {

        cout << "Area of Square: " << side \* side << endl;

    }

};

void calculateArea(Shape \*shape)

{

    shape->area();

    delete shape; // Deletes the shape and calls the correct destructor

}

int main()

{

    Shape \*shape = nullptr;

    int choice;

    do

    {

        cout << "-------- Calculate Area of Shapes ---------" << endl;

        cout << "1. Rectangle" << endl;

        cout << "2. Circle" << endl;

        cout << "3. Square" << endl;

        cout << "4. Exit" << endl;

        cin >> choice;

        switch (choice)

        {

        case 1:

        {

            float length, breadth;

            cout << "Enter length: ";

            cin >> length;

            cout << "Enter breadth: ";

            cin >> breadth;

            shape = new Rectangle(length, breadth); // Dynamically allocate Rectangle

            calculateArea(shape);

            break;

        }

        case 2:

        {

            float radius;

            cout << "Enter radius: ";

            cin >> radius;

            shape = new Circle(radius); // Dynamically allocate Circle

            calculateArea(shape);

            break;

        }

        case 3:

        {

            float side;

            cout << "Enter side length: ";

            cin >> side;

            shape = new Square(side); // Dynamically allocate Square

            calculateArea(shape);

            break;

        }

        case 4:

            cout << "Exiting..." << endl;

            return 0;

        default:

            cout << "Invalid choice!" << endl;

            return 1;

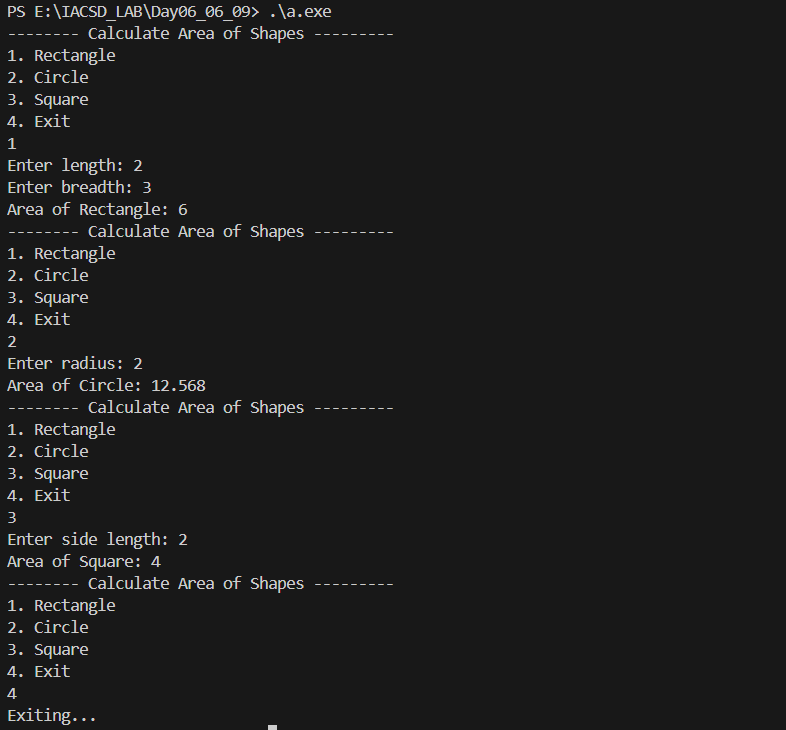
        }

    } while (choice != 4);

    return 0;

}

**Output :**

****

**Day8 – Inheritance Lab**

Animal Hierarchy:

Problem Statement: Create a hierarchy of animal classes. Start with a base class Animal and then create derived classes like Mammal, Bird, and Fish. Each of these derived classes should have specific properties and methods related to their respective categories of animals.

#include <iostream>

#include <string>

using namespace std;

class Animal {

protected:

    string name;

public:

    Animal(const string& name) : name(name) {}

    virtual void makeSound() const = 0;

    virtual ~Animal() {}

};

class Dog : public Animal {

public:

    Dog(const string& name) : Animal(name) {}

    void makeSound() const override {

        cout << "Woof!" << endl;

    }

};

int main() {

    string name;

    cout << "Enter the dog's name: ";

    getline(cin, name);

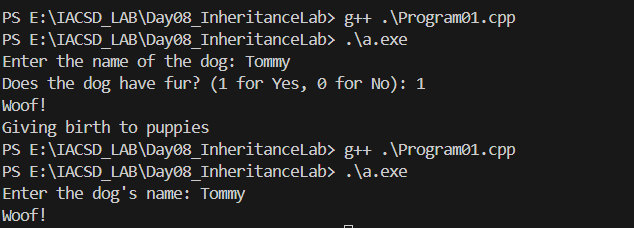
    Dog dog(name);

    dog.makeSound();

    return 0;

}

Output :



Shape Hierarchy:

Problem Statement: Design a hierarchy of shape classes. Begin with a base class Shape and then create derived classes like Circle, Rectangle, and Triangle. Each shape should have methods for calculating area and perimeter specific to its geometry.

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

    virtual double area() const = 0;

    virtual double perimeter() const = 0;

    virtual ~Shape() {}

};

class Circle : public Shape {

private:

    double radius;

public:

    Circle(double r) : radius(r) {}

    double area() const override {

        return 3.14 \* radius \* radius;

    }

    double perimeter() const override {

        return 2 \* 3.14 \* radius;

    }

};

class Rectangle : public Shape {

private:

    double width, height;

public:

    Rectangle(double w, double h) : width(w), height(h) {}

    double area() const override {

        return width \* height;

    }

    double perimeter() const override {

        return 2 \* (width + height);

    }

};

int main() {

    double radius, width, height;

    cout << "Enter the radius of the circle: ";

    cin >> radius;

    Circle circle(radius);

    cout << "Enter the width and height of the rectangle: ";

    cin >> width >> height;

    Rectangle rectangle(width, height);

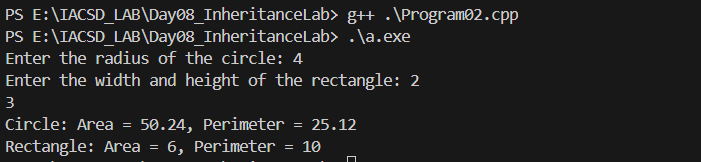
    cout << "Circle: Area = " << circle.area() << ", Perimeter = " << circle.perimeter() << endl;

    cout << "Rectangle: Area = " << rectangle.area() << ", Perimeter = " << rectangle.perimeter() << endl;

    return 0;

}

Output :



Employee Inheritance:

Problem Statement: Build a system for managing employees. Create a base class Employee with attributes such as name, employee ID, and salary. Then, derive classes like Manager and Developer, each with its own attributes and methods. Implement a common method, like calculate\_salary(), in the base class.

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

    string name;

    double salary;

public:

    Employee(const string& name, double salary)

        : name(name), salary(salary) {}

    virtual double calculateSalary() const = 0;

    virtual ~Employee() {}

};

class Manager : public Employee {

public:

    Manager(const string& name, double salary)

        : Employee(name, salary) {}

    double calculateSalary() const override {

        return salary + 5000; // Simple bonus for Manager

    }

};

class Developer : public Employee {

public:

    Developer(const string& name, double salary)

        : Employee(name, salary) {}

    double calculateSalary() const override {

        return salary + 2000; // Simple overtime pay for Developer

    }

};

int main() {

    string name;

    double salary;

    cout << "Enter Manager's name and salary: ";

    cin >> name >> salary;

    Manager manager(name, salary);

    cout << "Enter Developer's name and salary: ";

    cin >> name >> salary;

    Developer developer(name, salary);

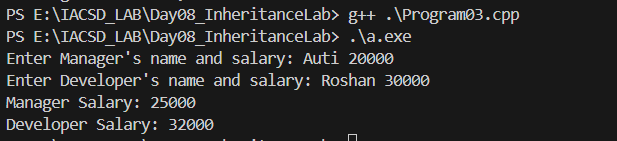
    cout << "Manager Salary: " << manager.calculateSalary() << endl;

    cout << "Developer Salary: " << developer.calculateSalary() << endl;

    return 0;

}

Output:



Vehicle Inheritance:

Problem Statement: Develop a class hierarchy for vehicles. Start with a base class Vehicle and create derived classes like Car, Motorcycle, and Truck. Each derived class should have unique properties like the number of wheels and specific methods like start\_engine().

#include <iostream>

#include <string>

using namespace std;

class Vehicle {

protected:

    int numberOfWheels;

public:

    Vehicle(int wheels) : numberOfWheels(wheels) {}

    virtual void startEngine() const = 0;

    virtual ~Vehicle() {}

};

class Car : public Vehicle {

public:

    Car() : Vehicle(4) {}

    void startEngine() const override {

        cout << "Car engine started" << endl;

    }

};

class Motorcycle : public Vehicle {

public:

    Motorcycle() : Vehicle(2) {}

    void startEngine() const override {

        cout << "Motorcycle engine started" << endl;

    }

};

int main() {

    Car car;

    Motorcycle motorcycle;

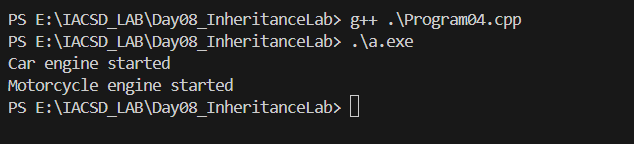
    car.startEngine();

    motorcycle.startEngine();

    return 0;

}

Output:



Bank Account Inheritance:

Problem Statement: Design a system for managing bank accounts. Create a base class BankAccount with attributes like account number and balance. Derive classes like SavingsAccount and CheckingAccount, each with specialized methods like withdraw() and calculate\_interest().

#include <iostream>

#include <string>

using namespace std;

class BankAccount {

protected:

    double balance;

public:

    BankAccount(double balance) : balance(balance) {}

    virtual void withdraw(double amount) = 0;

    virtual void calculateInterest() const = 0;

    virtual ~BankAccount() {}

};

class SavingsAccount : public BankAccount {

private:

    double interestRate;

public:

    SavingsAccount(double balance, double rate)

        : BankAccount(balance), interestRate(rate) {}

    void withdraw(double amount) override {

        if (amount <= balance) {

            balance -= amount;

        }

    }

    void calculateInterest() const override {

        double interest = balance \* interestRate;

        cout << "Interest: " << interest << endl;

    }

};

int main() {

    double balance, rate, amount;

    cout << "Enter Savings Account balance and interest rate: ";

    cin >> balance >> rate;

    SavingsAccount sa(balance, rate);

    cout << "Enter amount to withdraw: ";

    cin >> amount;

    sa.withdraw(amount);

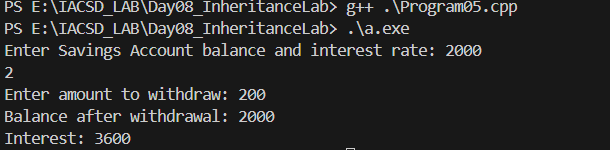
    cout << "Balance after withdrawal: " << balance << endl;

    sa.calculateInterest();

    return 0;

}

Output:



Geometric Shapes with Polymorphism:

Problem Statement: Extend the shape hierarchy example by implementing polymorphism. Define a base class Shape with methods to calculate area and perimeter. Then, create derived classes like Circle, Rectangle, and Triangle, each with its own implementation of these methods.

#include <iostream>

#include <cmath>

using namespace std;

class Shape {

public:

    virtual double area() const = 0;

    virtual double perimeter() const = 0;

    virtual ~Shape() {}

};

class Circle : public Shape {

private:

    double radius;

public:

    Circle(double r) : radius(r) {}

    double area() const override {

        return 3.14 \* radius \* radius;

    }

    double perimeter() const override {

        return 2 \* 3.14 \* radius;

    }

};

class Rectangle : public Shape {

private:

    double width, height;

public:

    Rectangle(double w, double h) : width(w), height(h) {}

    double area() const override {

        return width \* height;

    }

    double perimeter() const override {

        return 2 \* (width + height);

    }

};

class Triangle : public Shape {

private:

    double a, b, c;

public:

    Triangle(double x, double y, double z) : a(x), b(y), c(z) {}

    double area() const override {

        double s = (a + b + c) / 2;

        return sqrt(s \* (s - a) \* (s - b) \* (s - c));

    }

    double perimeter() const override {

        return a + b + c;

    }

};

int main() {

    double radius, width, height, a, b, c;

    cout << "Enter the radius of the circle: ";

    cin >> radius;

    Circle circle(radius);

    cout << "Enter the width and height of the rectangle: ";

    cin >> width >> height;

    Rectangle rectangle(width, height);

    cout << "Enter the sides of the triangle: ";

    cin >> a >> b >> c;

    Triangle triangle(a, b, c);

    Shape\* shapes[] = { &circle, &rectangle, &triangle };

    for (Shape\* shape : shapes) {

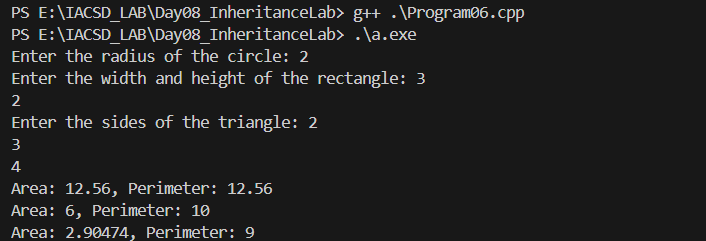
        cout << "Area: " << shape->area() << ", Perimeter: " << shape->perimeter() << endl;

    }

    return 0;

}

Output :



Person and Student Inheritance:

Problem Statement: Model a system for handling individuals and students within an educational institution. Create a base class Person with attributes like name and age. Derive a Student class with additional attributes like student ID and GPA, inheriting the common attributes from

the Person class.

#include <iostream>

#include <string>

using namespace std;

class Person {

protected:

    string name;

    int age;

public:

    Person(const string& name, int age) : name(name), age(age) {}

    virtual ~Person() {}

};

class Student : public Person {

private:

    string studentID;

    double GPA;

public:

    Student(const string& name, int age, const string& id, double gpa)

        : Person(name, age), studentID(id), GPA(gpa) {}

};

int main() {

    string name, studentID;

    int age;

    double GPA;

    cout << "Enter student's name, age, student ID, and GPA: ";

    cin >> name >> age >> studentID >> GPA;

    Student student(name, age, studentID, GPA);

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

    cout << "Age: " << age << endl;

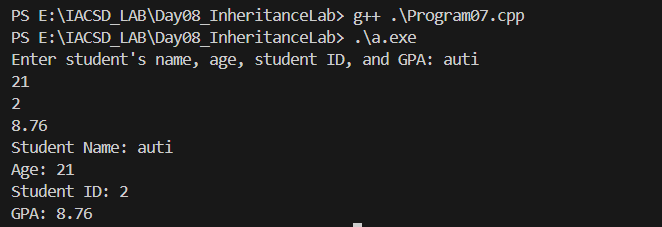
    cout << "Student ID: " << studentID << endl;

    cout << "GPA: " << GPA << endl;

    return 0;

}

Output :



Library Catalog with Books and Journals:

Problem Statement: Build a library catalog system. Create a base class LibraryItem with properties like title and author. Then, derive classes like Book and Journal, each with their unique properties. Implement methods to check out and return items in the derived classes.

#include <iostream>

#include <string>

using namespace std;

class LibraryItem {

protected:

    string title;

    string author;

public:

    LibraryItem(const string& title, const string& author)

        : title(title), author(author) {}

    virtual void checkOut() = 0;

    virtual void returnItem() = 0;

    virtual ~LibraryItem() {}

};

class Book : public LibraryItem {

public:

    Book(const string& title, const string& author)

        : LibraryItem(title, author) {}

    void checkOut() override {

        cout << "Book '" << title << "' checked out." << endl;

    }

    void returnItem() override {

        cout << "Book '" << title << "' returned." << endl;

    }

};

class Journal : public LibraryItem {

public:

    Journal(const string& title, const string& author)

        : LibraryItem(title, author) {}

    void checkOut() override {

        cout << "Journal '" << title << "' checked out." << endl;

    }

    void returnItem() override {

        cout << "Journal '" << title << "' returned." << endl;

    }

};

int main() {

    string title, author;

    cout << "Enter Book title and author: ";

    getline(cin, title);

    getline(cin, author);

    Book book(title, author);

    book.checkOut();

    book.returnItem();

    cout << "Enter Journal title and author: ";

    getline(cin, title);

    getline(cin, author);

    Journal journal(title, author);

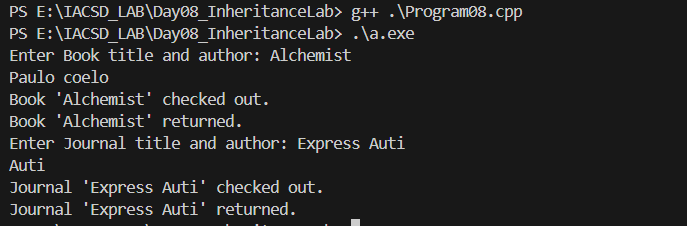
    journal.checkOut();

    journal.returnItem();

    return 0;

}

Output:



Shape Sorting with Interfaces:

Problem Statement: Implement a shape sorting program. Define a base class Shape with properties like area and perimeter. Create derived classes like Circle, Rectangle, and Triangle. Implement an interface Sortable with a method to compare shapes by area. Use this interface to sort a list of shapes.

#include <iostream>

#include <vector>

#include <algorithm>

#include <math.h>

using namespace std;

class Shape {

public:

    virtual double area() const = 0;

    virtual ~Shape() {}

};

class Circle : public Shape {

private:

    double radius;

public:

    Circle(double r) : radius(r) {}

    double area() const override {

        return 3.14 \* radius \* radius;

    }

};

class Rectangle : public Shape {

private:

    double width, height;

public:

    Rectangle(double w, double h) : width(w), height(h) {}

    double area() const override {

        return width \* height;

    }

};

class Triangle : public Shape {

private:

    double a, b, c;

public:

    Triangle(double x, double y, double z) : a(x), b(y), c(z) {}

    double area() const override {

        double s = (a + b + c) / 2;

        return sqrt(s \* (s - a) \* (s - b) \* (s - c));

    }

};

bool compareArea(const Shape\* s1, const Shape\* s2) {

    return s1->area() < s2->area();

}

int main() {

    vector<Shape\*> shapes;

    shapes.push\_back(new Circle(5));

    shapes.push\_back(new Rectangle(4, 6));

    shapes.push\_back(new Triangle(3, 4, 5));

    sort(shapes.begin(), shapes.end(), compareArea);

    for (const Shape\* shape : shapes) {

        cout << "Area: " << shape->area() << endl;

    }

    // Clean up

    for (Shape\* shape : shapes) {

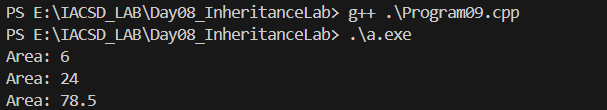
        delete shape;

    }

    return 0;

}

Output:



Employee Payroll System with Abstract Classes:

Problem Statement: Design an employee payroll system. Create an abstract class Employee with attributes like name and employee ID. Derive concrete classes like HourlyEmployee and SalariedEmployee. Define abstract methods for calculating pay in the base class and implement them in the derived classes.

#include <iostream>

#include <string>

using namespace std;

class Employee {

protected:

    string name;

    int employeeID;

public:

    Employee(const string& name, int id)

        : name(name), employeeID(id) {}

    virtual double calculatePay() const = 0;

    virtual ~Employee() {}

};

class HourlyEmployee : public Employee {

private:

    double hourlyRate;

    int hoursWorked;

public:

    HourlyEmployee(const string& name, int id, double rate, int hours)

        : Employee(name, id), hourlyRate(rate), hoursWorked(hours) {}

    double calculatePay() const override {

        return hourlyRate \* hoursWorked;

    }

};

class SalariedEmployee : public Employee {

private:

    double annualSalary;

public:

    SalariedEmployee(const string& name, int id, double salary)

        : Employee(name, id), annualSalary(salary) {}

    double calculatePay() const override {

        return annualSalary / 12; // Monthly salary

    }

};

int main() {

    string name;

    int id, hours;

    double rate, salary;

    cout << "Enter Hourly Employee's name, ID, hourly rate, and hours worked: ";

    cin >> name >> id >> rate >> hours;

    HourlyEmployee hourlyEmployee(name, id, rate, hours);

    cout << "Hourly Employee's Pay: " << hourlyEmployee.calculatePay() << endl;

    cout << "Enter Salaried Employee's name, ID, annual salary: ";

    cin >> name >> id >> salary;

    SalariedEmployee salariedEmployee(name, id, salary);

    cout << "Salaried Employee's Pay: " << salariedEmployee.calculatePay() << endl;

    return 0;

}

Output :

