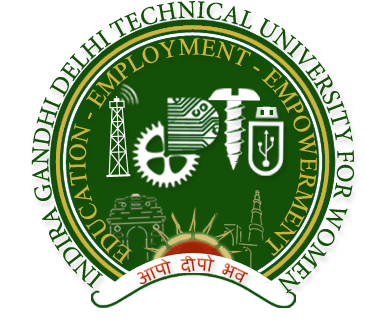
**Indira Gandhi Delhi Technical University for Women**

**(Established by Govt. of Delhi vide Act 09 of 2012)**

**(Formerly Indira Gandhi Institute of Technology)**

**Kashmere Gate, Delhi - 110006**



**LABORATORY FILE**

**for**

**Object Oriented Programming using C++**

**MCA-104**

**Submitted To:                                                   Submitted by:**

Ms. Ankita                                                          CHETNA

IT Department                                                   MCA 2nd Sem

IGDTUW, Delhi                                                  00604092019

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No** | **Topic** | **Date** | **Signature** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Classes, Objects & Functions**

**Ques 1**: Write a program to print “Hello World”.

**Solution 1**:

#include<iostream>

using namespace std;

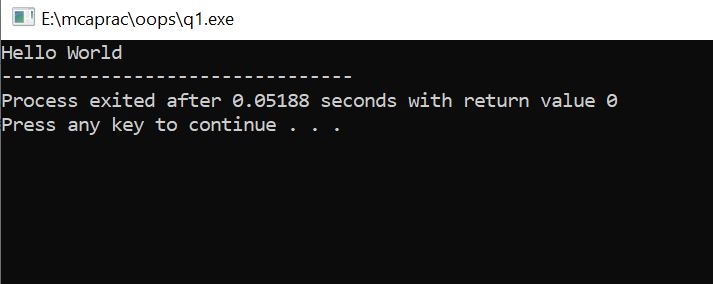
int main ()

{

    cout<<"Hello World";

    return 0;

}



**Ques 2:** Write a program to print the factorial of a number entered by the user.

**Solution 2:**

#include<iostream>

using namespace std;

//Function returning factorial of a given number

int fact (int n)

{

//base case

if(n==0)

return 1;

//induction hypothesis

int ans=fact(n-1);

//induction step

int myAns=n\*ans;

return myAns;

}

int main ()

{

int n, f;

cout<<"Enter a number: ";

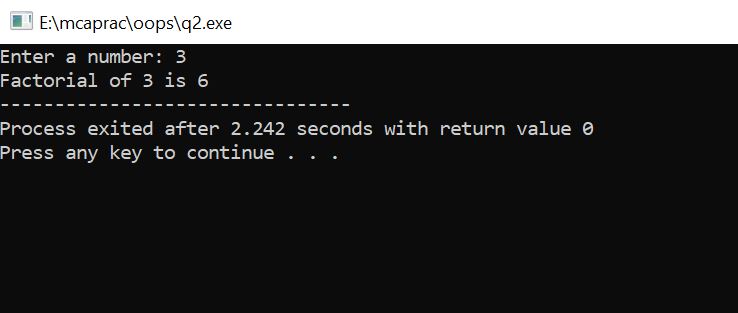
cin>>n;

f=fact(n);//Function call

     cout<<"Factorial of "<<n<<" is "<<f;

return 0;

}



**Ques 3:** Write a program to print Fibonacci series up to n.

**Solution 3:**

#include<iostream>

using namespace std;

int main ()

{

int n, c;

//Initialise first two numbers a and b

int a=0, b=1;

cout<<"Enter the number of terms: ";

cin>>n;

cout<<n<<" numbers of fibonacci series are: ";

cout<<a<<b;

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

{ c=a+b;

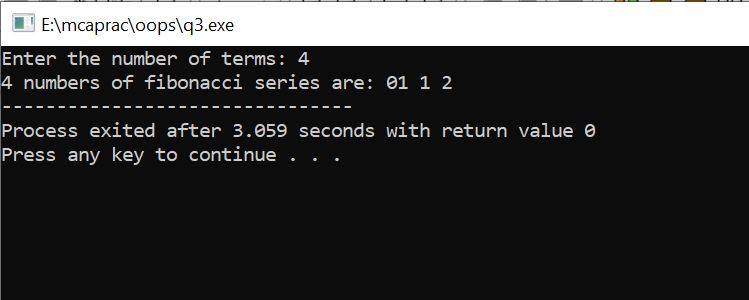
cout<<" "<<c;

a=b;

b=c;

}

return 0;}



**Ques 4:** Write a program to read an employee’s information from the user and print the same. Employee’s information shall include employee ID (*int*), employee name (*string*) and employee salary (*float*).

**Solution 4:**

#include<iostream>

using namespace std;

class employee

{

int emp\_num;

char emp\_name[20];

float sal;

float net\_sal;

float emp\_it;

public:

void get\_details();

void show\_emp\_details();

};

void employee :: get\_details()

{

cout<<"\nEnter employee number:\n";

cin>>emp\_num;

cout<<"\nEnter employee name:\n";

cin>>emp\_name;

cout<<"\nEnter employee salary:\n";

cin>>sal;

}

void employee :: show\_emp\_details()

{

cout<<"\n\n\nDetails of : "<<emp\_name;

cout<<"\n\nEmployee number: "<<emp\_num;

cout<<"\n\nEmployee Salary: "<<sal;

}

int main()

{

employee emp[10];

int i,num;

cout<<"\nEnter number of employee details\n";

cin>>num;

for(i=0;i<num;i++)

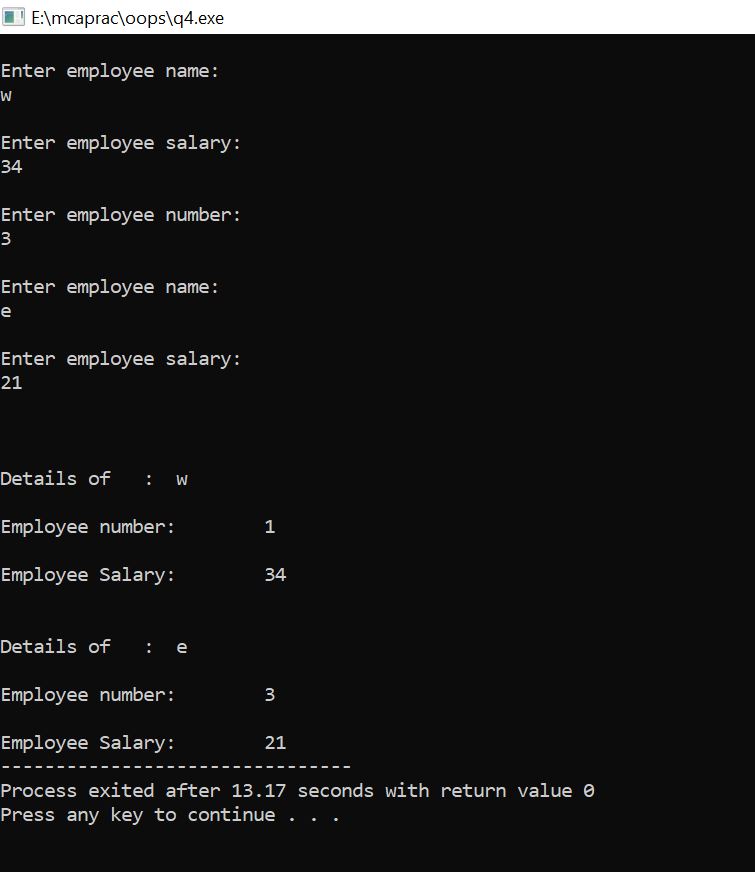
emp[i].get\_details();

for(i=0;i<num;i++)

emp[i].show\_emp\_details();

return 0;

}



**Ques 5:** Write a program to take two integer inputs and output their sum, difference, product and division (quotient and remainder) as result based on a third input. (operator).

**Solution 5:**

//input integers and output according to the operator

#include<iostream>

using namespace std;

int main ()

{

int a, b;

char op;

cout<<"\n Enter any two numbers:: ";

cin>>a>>b;

cout<<"Enter any operator of your choice:: ";

cin>>op;

switch(op){

case '+': cout<<"Sum is "<<(a+b) <<endl;

break;

case '-': cout<<"Difference is "<<(a-b) <<endl;

break;

case '/': cout<<"Quotient:: "<<(a/b);

cout<<" Remainder:: "<<(a%b) <<endl;

break;

case '\*': cout<<"Product is "<<(a\*b) <<endl;

break;

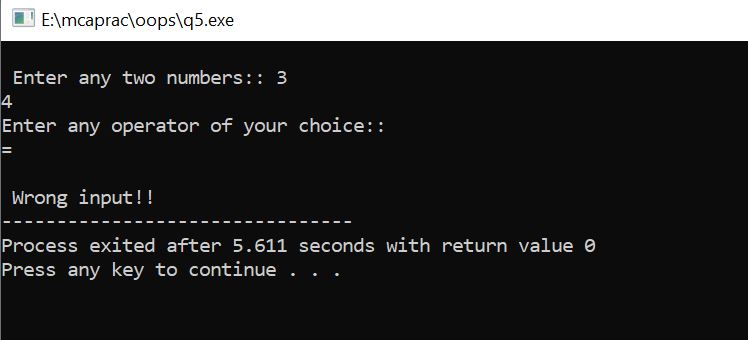
default: cout<<"\n Wrong input!!";

break;

}

return 0;

}



**Ques 6:** Write a program to perform the following conversions (and vice versa)

(a) Temperature in Celsius to Fahrenheit.

**Solution 6(a):**

#include<iostream>

using namespace std;

int main ()

{

float a, b;

int choice;

char ch;

cout<<"\*\*\*\* MENU \*\*\*\*\n";

cout<<"\n1. Convert from Celsius to Fahrenheit";

cout<<"\n2. Convert from Fahrenheit to Celsius\n";

do

{

cout<<"\n Enter your choice: ";

cin>>choice;//Entering choice

switch(choice)

{

case 1:  cout<<"Enter temperature in Celsius: ";

  cin>>a;

            b=(a\*9/5) + 32;

  cout<<"Temperature in Fahrenheit is: "<<b;

  break;

case 2:  cout<<"Enter temperature in Fahrenheit: ";

  cin>>a;

  b=((a-32) \*5)/9;

  cout<<"Temperature in Celsius is: "<<b;

  break;

default: cout<<"Wrong choice";

}

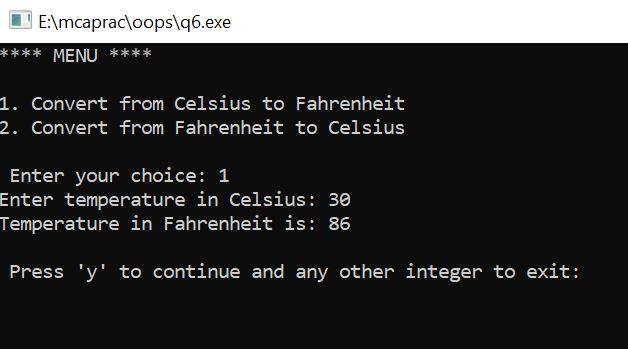
cout<<"\n\n Press 'y' to continue and any other integer to exit: ";

cin>>ch;

} while(ch=='y');

return 0;

}



(b) Height in Centimetres to Feet and Inches.

**Solution 6(b):**

#include<iostream>

using namespace std;

int main ()

{

float a, b;

int choice;

char ch;

cout<<"\*\*\*\* MENU \*\*\*\*\n";

cout<<"\n1. Convert from Centimeters to Feet";

cout<<"\n2. Convert from Centimeters to Inches";

cout<<"\n3. Convert from Feet to Centimeters";

cout<<"\n4. Convert from Inches to Centimeters\n";

do

{

cout<<"\n Enter your choice(1-4): ";

cin>>choice;//Entering choice

switch(choice)

{

case 1:   cout<<"Enter height (in cm): ";

  //cm refers to Centimeter

  cin>>a;

  b=a/30.48;

  cout<<"Height in Feet is: "<<b;

  break;

case 2:   cout<<"Enter height (in cm): ";

  //cm refers to Centimeter

  cin>>a;

  b=a/ 2.54;

  cout<<"Height in Inches is: "<<b;

  break;

case 3:  cout<<"Enter height (in Feet): ";

  cin>>a;

  b=a\*30.48;

  cout<<"Height in cm is: "<<b;

  break;

case 4:  cout<<"Enter height (in Inches): ";

  cin>>a;

  b=a\*2.54;

  cout<<"Height in cm is: "<<b;

  break;

default: cout<<"Wrong choice";

}

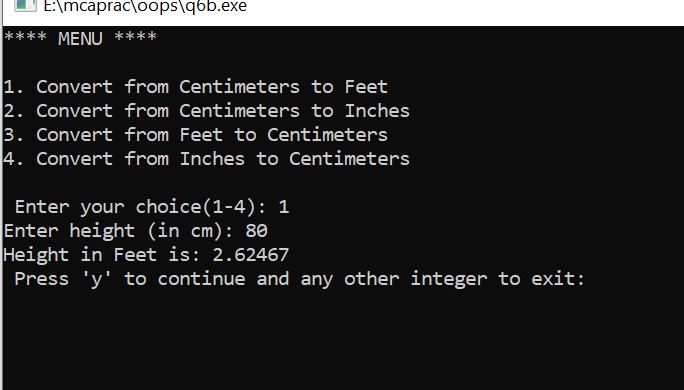
cout<<"\n Press 'y' to continue and any other integer to exit: ";

cin>>ch;

} while(ch=='y');

return 0;

}



**Ques 7:** A perfect number is the one whose divisors add up to the number. For example, 6 is a perfect number because 6=1+2+3. Write a program that prints all perfect numbers from 1 till 10000.

**Solution 7:**

//program to find perfect numbers between 1-1000

#include<iostream>

using namespace std;

int main ()

{

int j=1;

//Initializing number to be checked as 1

cout<<"Perfect numbers between 1-10000 are as follows:\n";

do {

int sum=0;

         for (int i=1; i<j; i++)

         {

             if (j%i ==0)

                 sum+=i;

}

//checking sum equal or not

         if(sum==j)

             cout<<j<<" ";

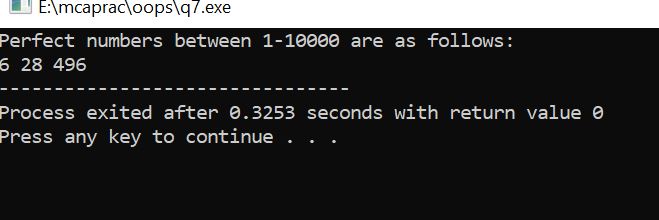
         j++;

} while(j<=10000);

//end of while loop

return 0;

}



**Ques 8:** Write a program to swap two numbers: using call by value and call by reference & pointer.

**Solution 8:**

//swapping using call by value and call by reference

#include<iostream>

using namespace std;

int swap (int a, int b)

{

int t=a;

a=b;

b=t;

cout<<a<<" "<<b;

}

int swap1(int \*a, int \*b)

{

int t=\*a;

\*a=\*b;

\*b=t;

cout<<\*a<<" "<<\*b;

}

int main ()

{

int a, b;

cout<<"Enter a and b::";

cin>>a>>b;

cout<<"Value of a and b before swapping:: "<<a<<" "<<b<<endl;

cout<<"\n After swapping: ";

cout<<"\n Values of a and b using call by value:: ";

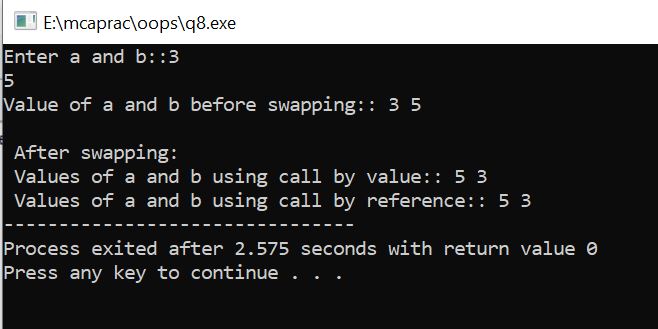
swap (a, b);

cout<<"\n Values of a and b using call by reference:: ";

swap1(&a, &b);

return 0;

}



**Ques 9:** Write a program that calculates area and perimeter of the following geometric figures. Your program should use *function overloading* and each function should take as inputs the required arguments (without constraining the user) and return both area and perimeter.

(a) Square

(b) Rectangle

**Solution 9:**

#include<iostream>

using namespace std;

float area (float side) {

return side\*side;

}

float area (float len, float bre) {

return len\*bre;

}

float peri (float side) {

return 4\*side;

}

float peri (float len, float bre) {

return 2\*(len+ bre);

}

int main ()

{

float s, l, b;

  cout<<"Enter side of square: ";

  cin>>s;

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

cin>>l>>b;

  cout<<"\n Area of Square = "<<area(s);

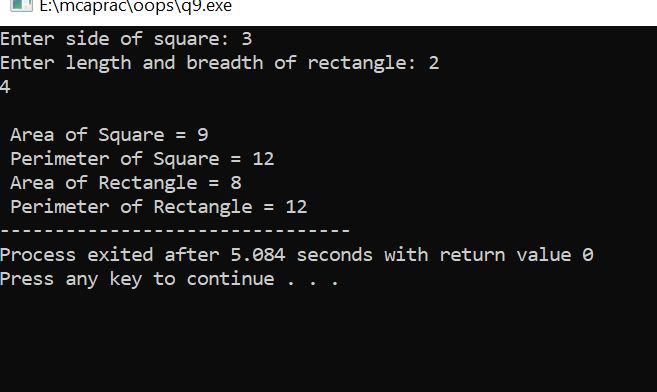
cout<<"\n Perimeter of Square = "<<peri(s);

  cout<<"\n Area of Rectangle = "<<area (l, b);

cout<<"\n Perimeter of Rectangle = "<<peri (l, b);

return 0;

}



**Ques 10:** Write a program with *recursive functions* that perform the following:

(a) Reverse an input string

**Solution 10(a):**

#include <iostream>

#include <cstring>

using namespace std;

//recursive function for reversing the string.

void reverse (char \*x, int begin, int end)

{

    char c;

    if (begin >= end)

       return;

c = \*(x+ begin);

    \*(x+ begin) = \*(x+ end);

    \*(x+ end) = c;

    reverse (x, ++begin, --end);

}

int main ()

{

    char a[100];

cout<<"Enter a string: ";

    cin>>a;

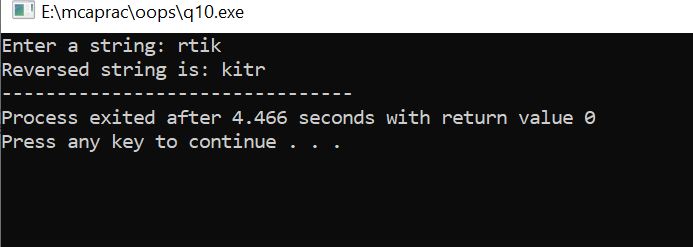
//function call

reverse (a, 0, strlen(a)-1);

cout<<"Reversed string is: "<<a;

return 0;

}



(b) Check if an input string is palindrome or not.

**Solution 10(b):**

#include <iostream>

#include <string.h>

using namespace std;

//recursive function for palindrome check.

bool isPalindrome (const string &str, int start, int end)

{

     if (start >= end)

         return true;

     if (str[start] != str[end])

         return false;

     return isPalindrome (str, ++start, --end);

}

int main ()

{

  char str[20];

     int flag = 0;

     cout<<"Enter a string: ";

     cin>>str;

//to find length of the string

int len = strlen(str)-1;

if (isPalindrome (str, 0, len)) //function call

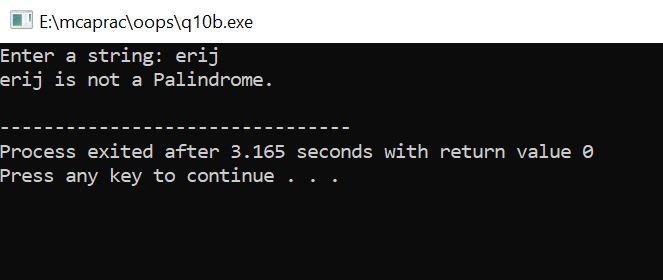
         cout << str << " is a Palindrome." << endl;

     else

         cout << str << " is not a Palindrome." << endl;

     return 0;

}



**Ques 11:** What are the different ways in which objects of a class can be created? Explain with an illustrative program for each

**Solution 11:**

1. Using *new* keyword

#include <iostream>

#include<stdlib.h>

using namespace std;

class Shape{

public:

    Shape() //default

    {

        cout<<"Default Constructor"<<endl;

    }

    Shape(const Shape& obj) //copy constructor

    {

        cout<<"Copy Constructor"<<endl;

    }

    void \* operator new(size\_t size) //operator overloading

    {

        cout<<"New ";

        void \* storage=malloc(size);

        if(NULL == storage)

        {

            cout<<"Allocation fail : no free memory";

        }

    return storage;

}

};

int main()

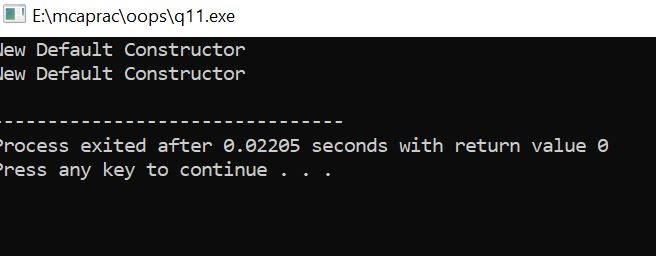
{

Shape \*ptrShape=new Shape();

Shape \*ptrShapeOver= new Shape();

return 0;

}



1. Using the class name

#include <iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Shape{

public:

    Shape() //default

    {

        cout<<"Default Constructor"<<endl;

    }

    Shape(const Shape& obj) //copy constructor

    {

        cout<<"Copy Constructor"<<endl;

    }

    void \* operator new(size\_t size) //operator overloading

    {

        cout<<"New ";

        void \* storage=malloc(size);

        if(NULL == storage)

        {

            cout<<"Allocation fail : no free memory";

        }

    return storage;

}

};

int main()

{

Shape obj;

Shape obj2=obj;

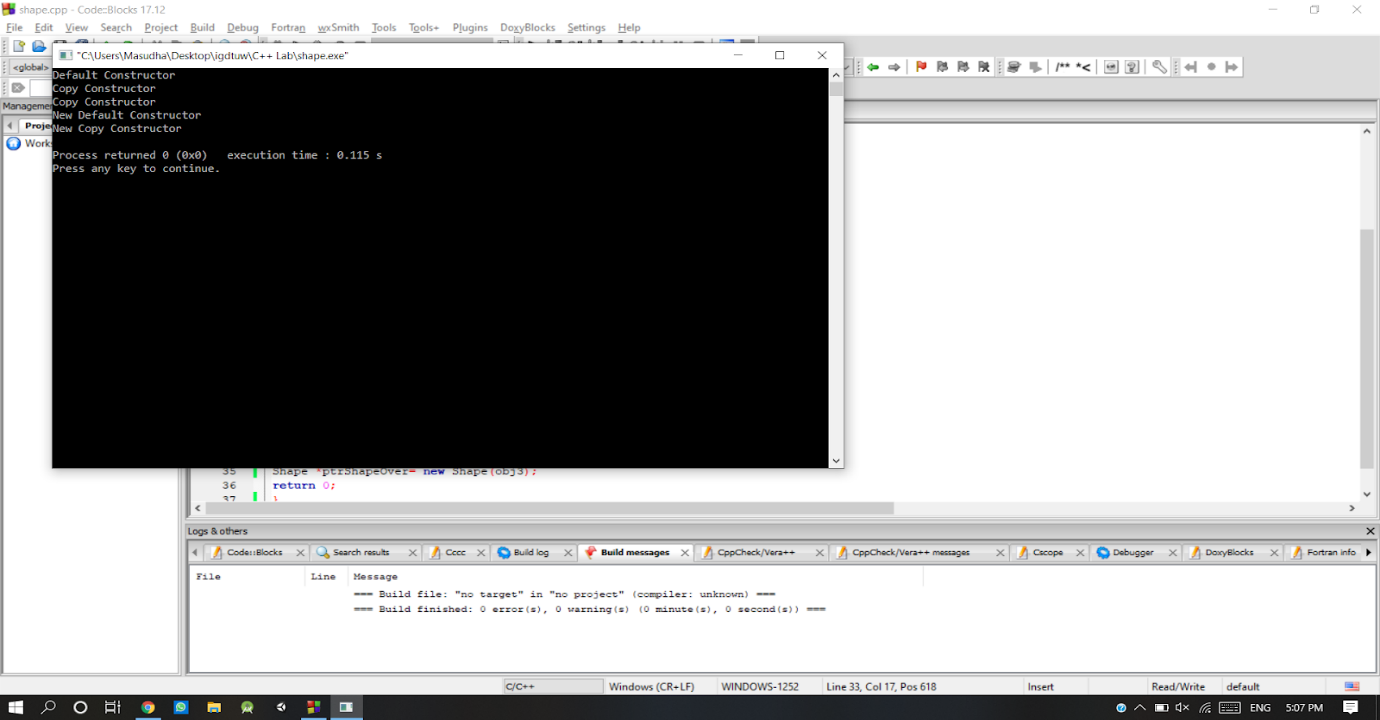
Shape obj3(obj);

Shape \*ptrShape=new Shape();

Shape \*ptrShapeOver= new Shape(obj3);

return 0;

}



**Ques 13:** Write a program that illustrates:

(a) Use of friend function

**Solution 13(a):**

#include <iostream>

using namespace std;

class Number

{private:

         int a;

     public:

         void getNum(int x);

         friend void printNum(Number NUM);

};

 void Number::getNum(int x)

{ a=x; }

void printNum(Number NUM)

{ cout << "Value of a (private data member of class Number): " << NUM.a;

 }

 int main()

{

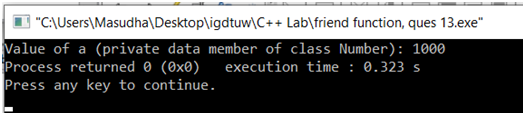
Number nObj; //Object declaration

nObj.getNum(1000);

printNum(nObj);

return 0;

}



(b)Member function as a friend function

**Solution 13(b):**

#include<iostream>

using namespace std;

class B; //declare class B

class A {

private:

int a;

public:

//constructor

A(int a)

{

     this->a = a;

}

friend int max(A a, B b);

};

class B {

private:

int b;

public:

//constructor

B(int b) {

     this->b = b;

}

 friend int max(A a, B b);};

int max(A a, B b) {

return (a.a > b.b ? a.a : b.b);}

int main() {

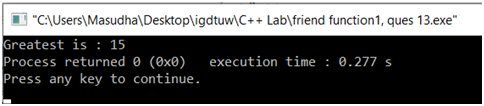
A a(10);

B b(15);

cout << "Greatest is : " << max(a, b);

return 0;

}



**Ques 14:** Account database is maintaining details of customers. Design suitable class and write member functions which provide the following functions:

(a)Insert details of customer

(b) Search an existing customer w.r.t. account no.

(c) Modify the details of an existing customer

**Solution 14:**

#include<iostream>

using namespace std;

class account

{

public:

     int   cust\_num;

     char  cust\_name[20];

     long long int phone;

                           void insert\_details();

                           void search\_details();

                           void modify\_details();

                           void show\_details();

};

void account :: insert\_details()

{

            cout<<"\nEnter customer number:\n";

            cin>>cust\_num;

            cout<<"\nEnter customer name:\n";

            cin>>cust\_name;

            cout<<"\nEnter customer phone number:\n";

            cin>>phone;

}

void account :: show\_details()

{

    cout<<"\nThe customer details are as follows\n";

    cout<<cust\_num<<"\t"<<cust\_name<<"\t"<<phone;

}

void account :: search\_details()

{

            int search\_customer;

            cout<<"\nEnter the customer number that you want to search\n";

            cin>>search\_customer;

            if(search\_customer==cust\_num)

{

        cout<<"\nCustomer found\n";

}

}

void account:: modify\_details()

{

int search\_customer;

            cout<<"Enter the customer number that you want to search";

            cin>>search\_customer;

            if(search\_customer==cust\_num)

{

        cout<<"Enter the choice number, what you want to change\n";

        cout<<"1. Customer Name\n";

        cout<<"2. Phone Number\n";

     int choice=-1;

        cin>>choice;

     switch(choice)

     {

         case 1:

                cout<<"Enter new name";

                cin>>cust\_name;

             break;

         case 2:

                cout<<"Enter new phone number";

                cin>>phone;

             break;

         default:

                cout<<"Invalid choice.";

             break;

     }

}

}

int main()

{

            account a[10];

            int i,num;

  cout<<"\nEnter number of employee details\n";

            cin>>num;

  for(i=0;i<num;i++)

                           a[i].insert\_details();

     for(i=0;i<num;i++)

                           a[i].show\_details();

    for(i=0;i<num;i++)

{

        a[i].search\_details();

                           break;

}

    for(i=0;i<num;i++)

{

                           a[i].modify\_details();

                           break;

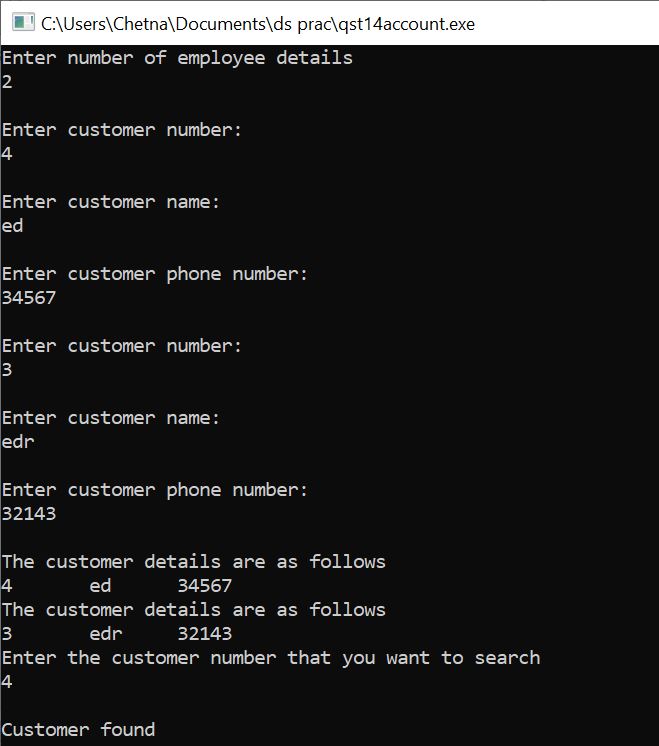
}

for(i=0;i<num;i++)

                           a[i].show\_details();

return 0;

}



**Operator Overloading**

**Ques 15:** Write a program which provides concrete representation for the concept of complex numbers. Using *operator overloading,* perform the operations of addition and multiplication of two given complex numbers. Implement operator overloading.

**Solution 15:**

#include<iostream>

using namespace std;

class complex

{

int i,r;

public:

void read()

{

cout<<"\nEnter Real Part:";

cin>>r;

cout<<"Enter Imaginary Part:";

cin>>i;

}

void display()

{

cout<<"\n= "<<r<<"+"<<i<<"i";

}

complex operator+(complex a2)

{

complex a;

a.r=r+a2.r;

a.i=i+a2.i;

return a;

}

complex operator-(complex a2)

{

complex a;

a.r=r-a2.r;

a.i=i-a2.i;

return a;

}

complex operator\*(complex a2)

{

complex a;

a.r=(r\*a2.r)-(i\*a2.i);

a.i=(r\*a2.i)+(i\*a2.r);

return a;

}

complex operator/(complex a2)

{

complex a;

a.r=((r\*a2.r)+(i\*a2.i))/((a2.r\*a2.r)+(a2.i\*a2.i));

a.i=((i\*a2.r)-(r\*a2.i))/((a2.r\*a2.r)+(a2.i\*a2.i));

return a;

}

};

int main()

{

int ch;

complex a,b,c;

do

{

cout<<"\n1.Addition 2.Substraction";

cout<<" 3.Mulitplication 4.Division 5.Exit\n";

cout<<"\nEnter the choice :";

cin>>ch;

switch(ch)

{

case 1:

cout<<"\nEnter The First Complex Number:";

a.read();

a.display();

cout<<"\nEnter The Second Complex Number:";

b.read();

b.display();

c=a+b;

c.display();

break;

case 2:

cout<<"\nEnter The First Complex Number:";

a.read();

a.display();

cout<<"\nEnter The Second Complex Number:";

b.read();

b.display();

c=b-a;

c.display();

break;

case 3:

cout<<"\nEnter The First Complex Number:";

a.read();

a.display();

cout<<"\nEnter The Second Complex Number:";

b.read();

b.display();

c=a\*b;

c.display();

break;

case 4:

cout<<"\nEnter The First Complex Number:";

a.read();

a.display();

cout<<"\nEnter The Second Complex Number:";

b.read();

b.display();

c=a/b;

c.display();

break;

}

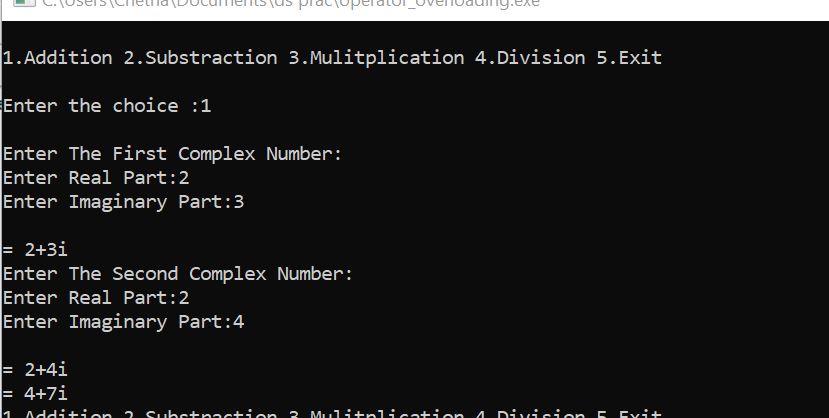
}while(ch!=5);

getch();

return 0;

}

**Output:**



**Ques 16:** Extend the concept of complex number by providing provisions for unary operators + and – which increments and decrements both real and imaginary part of complex numbers by 1, respectively. Implement operator overloading.

**Solution 16:**

**#**include<iostream>

#include<conio.h>

using namespace std;

class complex{

int a,b,c;

public:

complex(){}

void getvalue(){

cout<<"Enter the Two Numbers:";

cin>>a>>b; }

void operator++(){

a=++a;

b=++b;

}

void operator--() {

a=--a;

b=--b;

}

void display()

{ cout<<a<<"+\t"<<b<<"i"<<endl; } };

int main(){

complex obj;

obj.getvalue();

++obj;

cout<<"Increment Complex Number\n";

obj.display();

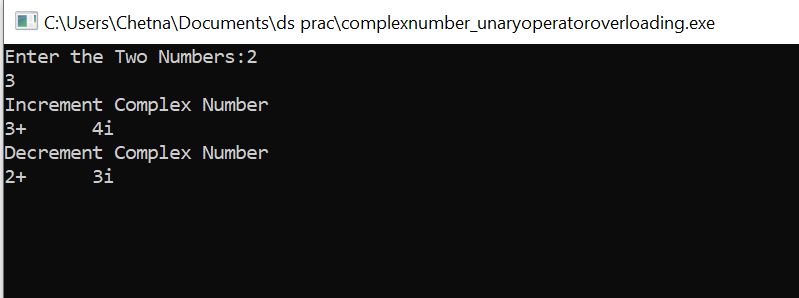
--obj;

cout<<"Decrement Complex Number\n";

obj.display();

getch();}

**output:**



**Ques 17:** Write a program to overload the following operators:

1. Subscript operator

**Solution17(a):**

#include<iostream>

using namespace std;

const int SIZE=5;

class arraytype

{

    int a[SIZE];

public:

    arraytype()

    {

        int i;

        for(i=0; i<SIZE; i++)

        {

            a[i]=i;

        }

    }

    int operator[](int i)

    {

        return a[i];

    }

};

int main()

{

    arraytype ob;

    int i;

    for(i=0; i<SIZE; i++)

    {

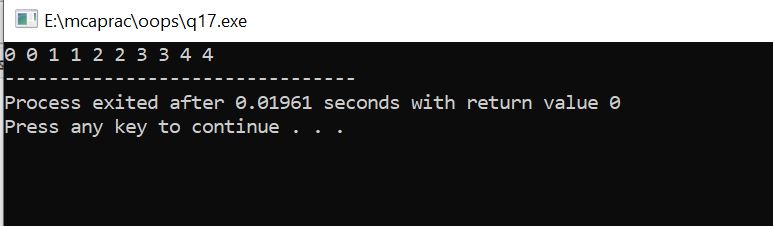
        cout<<ob.operator[](i)<<" ";

        cout<<ob[i]<<" ";

    }

    return 0;

}



1. Function call operator

**Solution17(b):**

#include <iostream>

using namespace std;

class Distance {

   private:

      int feet;

      int inches;

   public:

      Distance() {

         feet = 0;

         inches = 0;

      }

      Distance(int f, int i) {

         feet = f;

         inches = i;

      }

      // overload function call

      Distance operator()(int a, int b, int c) {

         Distance D;

         D.feet = a + c + 10;

         D.inches = b + c + 100 ;

         return D;

      }

      void displayDistance() {

         cout << "F: " << feet << " I:" << inches << endl;

      }

};

int main() {

   Distance D1(11, 10), D2;

   cout << "First Distance : ";

   D1.displayDistance();

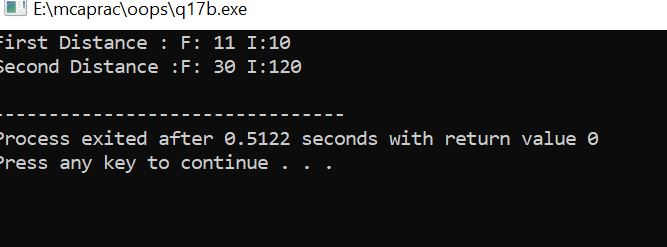
   D2 = D1(10, 10, 10); // invoke operator()

   cout << "Second Distance :";

   D2.displayDistance();

   return 0;

}



1. Assignment operator

**Solution17(c):**

#include<iostream>

using namespace std;

class coord

{

    int x, y;

public:

    coord()

    {

        x=0; y=0;

    }

    coord(int i, int j)

    {

        x=i; y=j;

    }

    int operator==(coord ob2);

};

int coord::operator==(coord ob2)

{

    return((x==ob2.x)&&(y==ob2.y));

}

int main()

{

    coord o1(10,10), o2(5,3), o3(10,10), o4(0,0);

    if(o1==o2)

        cout<<"O1 is same as O2"<<endl;

    else

        cout<<"Not same"<<endl;

    if(o1==o3)

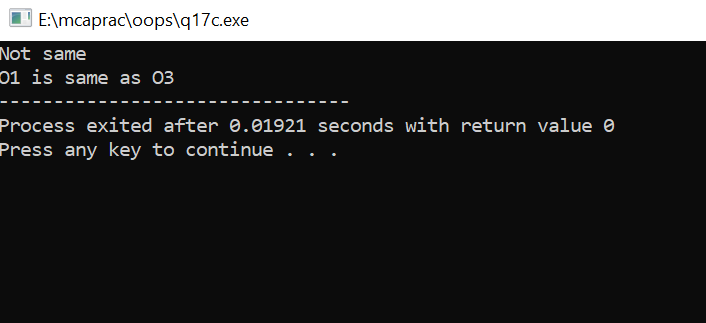
        cout<<"O1 is same as O3";

    else

        cout<<"Not same";

    return 0;

}



**Inheritance**

**Ques 18:** Write a C++ program that represents a Vehicle-Car and Vehicle-Motorcycle relationship C++ program should provide following operations.

1. Add a vehicle (car/motorcycle)
2. Design the vehicles (including both cars and motorcycles) with  respect to their registration ID (alpha-numeric)

**Solution 18:**

#include<iostream>

#include<string>

using namespace std;

class Vehicle

{

public:

    char vehiclename[10];

    int registrationID;

    char model[10];

    int year;

    void addVehicle()

    {

        cout<<"Enter the registration ID: ";

        cin>>registrationID;

        cout<<"Enter the vehicle name: ";

        cin>>vehiclename;

        cout<<"Enter the model: ";

        cin>>model;

        cout<<"Enter the year: ";

        cin>>year;

    }

    void display()

    {

        cout<<"Registration ID: "<<registrationID<<" Vehicle is: "<<vehiclename<<" Model: "<<model<<" Year: "<<year<<endl;

    }

};

class Car : public Vehicle

{

    public:

    void carDetails()

    {

        cout<<"Car Details are as follows"<<endl;

    }

};

class Motorcycle : public Vehicle

{

    public:

    void motorcycleDetails()

    {

        cout<<"\*\*\*\*\*\*\*\*\*\*List of Motorcycles\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

    }

};

int main()

{

    cout<<"\*\*\*\*\*\*\*\*\*ADD VEHICLE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

    string answer;

    cout<<"Do you want to add a car or motorcycle?"<<endl;

    getline(cin, answer);

    if(answer == "car")

    {

        int n;

        cout<<"How many cars you want to add? ";

        cin>>n;

        //car

        Car c[n];

        for(int i=0; i<n; i++)

        {

            c[i].addVehicle();

        }

        for(int i=0; i<n; i++)

        {

            c[i].display();

        }

    }

    else

    {

       int n;

        cout<<"How many motorcycles you want to add? ";

        cin>>n;

        //car

        Motorcycle m[n];

        for(int i=0; i<n; i++)

        {

            m[i].addVehicle();

        }

        for(int i=0; i<n; i++)

        {

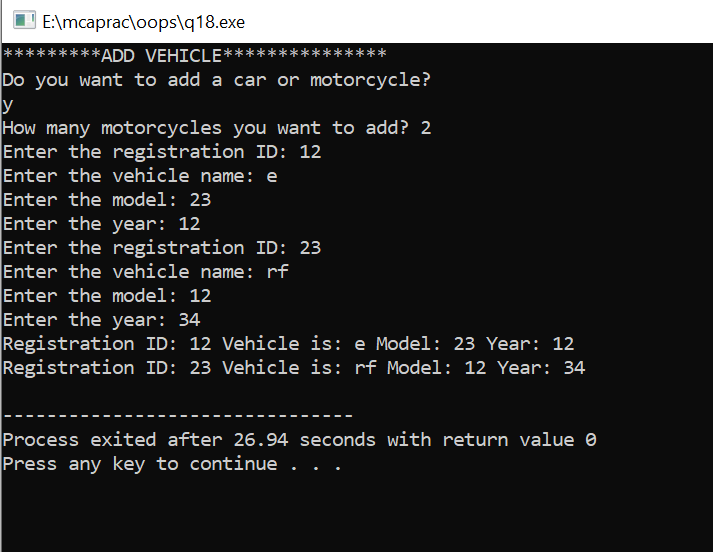
            m[i].display();

        }

    }

   return 0;

}



**Ques 19:** Design a single C++ program illustrating the following concept of inheritance:

1. Single Inheritance

**Solution 19(a):**

#include <iostream>

using namespace std;

class base    //single base class

{

   public:

     int x;

   void getdata()

   {

     cout << "Enter the value of x = "; cin >> x;

   }

 };

class derive : public base    //single derived class

{

   private:

    int y;

   public:

   void readdata()

   {

     cout << "Enter the value of y = "; cin >> y;

   }

   void product()

   {

     cout << "Product = " << x \* y;

   }

 };

 int main()

 {

    derive a;     //object of derived class

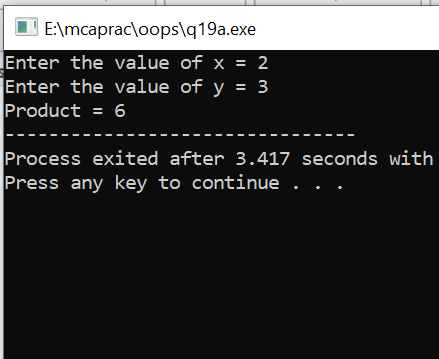
    a.getdata();

    a.readdata();

    a.product();

    return 0;

 }



1. Multiple Inheritance

**Solution 19(b):**

#include<iostream>

using namespace std;

class A

{

  public:

  int x;

  void getx()

    {

      cout << "enter value of x: "; cin >> x;

    }

};

class B

{

  public:

  int y;

  void gety()

  {

      cout << "enter value of y: "; cin >> y;

  }

};

class C : public A, public B   //C is derived from class A and class B

{

  public:

  void sum()

  {

      cout << "Sum = " << x + y;

  }

};

int main()

{

  C obj1; //object of derived class C

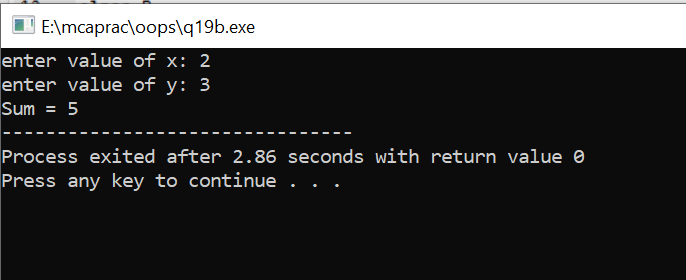
  obj1.getx();

  obj1.gety();

  obj1.sum();

  return 0;

}



1. Multi-level Inheritance

**Solution 19(c):**

#include <iostream>

using namespace std;

class base //single base class

{

  public:

  int x;

  void getdata()

  {

     cout << "Enter value of x= "; cin >> x;

  }

};

class derive1 : public base // derived class from base class

{

  public:

  int y;

  void readdata()

  {

      cout << "\nEnter value of y= "; cin >> y;

  }

};

class derive2 : public derive1   // derived from class derive1

{

  private:

  int z;

  public:

  void indata()

  {

     cout << "\nEnter value of z= "; cin >> z;

  }

  void product()

  {

      cout << "\nProduct= " << x \* y \* z;

  }

};

int main()

{

     derive2 a;      //object of derived class

     a.getdata();

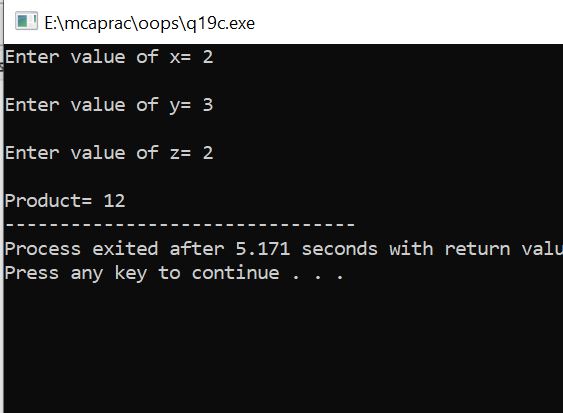
     a.readdata();

     a.indata();

     a.product();

     return 0;

}



**Ques 20:** Design a single C++ program illustrating the following concept of inheritance:

1. Public Derivation

**Solution 20 (a):**

#include <iostream>

using namespace std;

// Base class

class Shape {

   public:

      void setWidth(int w) {

         width = w;

      }

      void setHeight(int h) {

         height = h;

      }

   protected:

      int width;

      int height;

};

// Derived class

class Rectangle: public Shape {

   public:

      int getArea() {

         return (width \* height);

      }

};

int main(void) {

   Rectangle Rect;

   Rect.setWidth(5);

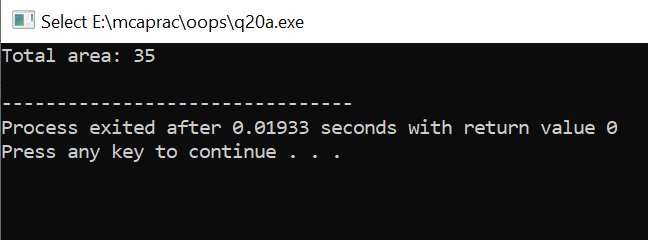
   Rect.setHeight(7);

   // Print the area of the object.

   cout << "Total area: " << Rect.getArea() << endl;

   return 0;

}



1. Private Derivation

**Solution 20 (b):**

#include <iostream>

using namespace std;

class Parent{

public:

  void parentMethod( void )

{ cout<<"Inside parent method"<<endl;}

};

class Child : private Parent{

public:

  void childMethod( void){

    cout<<"Inside child method"<<endl;

    parentMethod();

  }

};

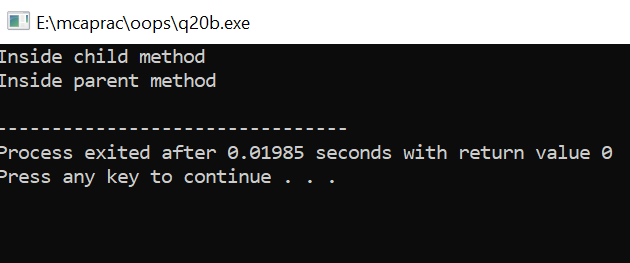
int main( void ){

  Child C;

  C.childMethod();

  return 0;

}



1. Protected Derivation

**Solution 20 (c):**

#include <iostream>

using namespace std;

class GrandParent{

public:

  void grandParentMethod( void )

{ cout<<"Method in the grand parent class"<<endl; }

};

class Parent : protected GrandParent{

public:

  void parentMethod( void ){ cout<<"Method in the parent class"<<endl; }

};

class Child: protected Parent{

public:

  void

  childMethod( void ){

    cout<<"Method in the child class"<<endl;

    parentMethod();

    grandParentMethod();

  }

};

int

main( void ){

  Child C;

  C.childMethod();

  return 0;

}

