M.Sc. (Five Year Integrated) in Computer Science (Artificial Intelligence & Data Science)

First Semester

Laboratory Record

21-805-0107: C++ PROGRAMMING LAB

Submitted in partial fulfillment
of the requirements for the award of degree in
Master of Science (Five Year Integrated)
in Computer Science (Artificial Intelligence & Data Science) of
Cochin University of Science and Technology (CUSAT)
Kochi

Submitted by

ABDUL HAKKEEM P A
(80521001)

DEPARTMENT OF COMPUTER SCIENCE
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)
KOCHI-682022

MARCH 2022

DEPARTMENT OF COMPUTER SCIENCE

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)

KOCHI, KERALA-682022

This is to certify that the software laboratory record for ***21-805-0107: C++ Programming Lab*** *is a record of work carried out by* ***Abdul Hakkeem P A (80521001)****, in partial fulfillment of the requirements for the award of degree in* ***Master of Science (Five Year Integrated)*** *in* ***Computer Science (Artificial Intelligence & Data Science)*** *of Cochin University of Science and Technology (CUSAT), Kochi. The lab record has been approved as it satisfies the academic requirements in respect of the first semester laboratory prescribed for the Master of Science (Five Year Integrated) in Computer Science degree.*

Faculty Member in-charge

Dr. Madhu S. Nair
Professor
Department of Computer Science
CUSAT

Dr. Philip Samuel
Professor and Head
Department of Computer Science
CUSAT

Table of Contents

STUDENT GRADES

AIM

Write a C++ program to calculate the grades of a list of students with attributes (Name, Roll no, Marks of 3 subjects) using class with member functions input(), calcGrade(), display().

PROGRAM

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

class student{
    std::string name;
    int roll_no;
    float mark1,mark2,mark3,totalMarks;
    char grade;

    public:
        void input();
        char calcGrade(float,float,float);
        void display();
};

void student::input(){
    std::cout<<"Enter the Student Name : ";
    std::cin>>name;
    std::cout<<"Enter the Roll No : ";
    std::cin>>roll_no;
    std::cout<<"Enter the Marks for Subject 1 : ";
    std::cin>>mark1;
    std::cout<<"Enter the Marks for Subject 2 : ";
    std::cin>>mark2;
    std::cout<<"Enter the Marks for Subject 3 : ";
    std::cin>>mark3;
}

char student::calcGrade(float mark1,float mark2,float mark3){
    totalMarks = (mark1+mark2+mark3)/3;
    if(totalMarks>=90){
        grade='A';
    }
```

```
    else if(totalMarks>=80 and totalMarks<90){
        grade='B';
    }
    else if(totalMarks>=70 and totalMarks<80){
        grade='C';
    }
    else if(totalMarks>=60 and totalMarks<70){
        grade='D';
    }
    else if(totalMarks>=50 and totalMarks<60){
        grade='E';
    }
    else{
        grade='F';
    }
    return grade;
}

void student::display(){
    std::cout<<endl;
    std::cout<<"Student : "<<name;
    std::cout<<"\nThe Grade is "<<calcGrade(mark1,mark2,mark3)<<std::endl;
}

int main() {
    int exitOption,choice;
    std::cout<<"1.Start\n2.Quit"<<endl;
    std::cin>>exitOption;
    if (exitOption==2) {
        return 0;
    }
    do{
        student Student1;
        Student1.input();
        Student1.display();
        std::cout<<"\nDo you want to continue \n1.Continue\n2.Quit"<<std::endl;
        std::cin>>choice;
    } while(choice == 1);
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
Enter the Student Name : Hakkeem
Enter the Roll No : 1
Enter the Marks for Subject 1 : 85
Enter the Marks for Subject 2 : 75
Enter the Marks for Subject 3 : 55

Student : Hakkeem
The Grade is C
```

OVERLOADING FUNCTIONS TO FIND AREA

AIM

Write a C++ prgram to calculate the area of different shapes like Rectangle, Square etc (at least 5 shapes) using overloaded function area().

PROGRAM

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

void area(int length,int breadth){
    int area;
    area=length*breadth;
    std::cout<<"The Area of the Rectangle is "<<area<<std::endl;
}

void area(int side){
    int area;
    area=side*side;
    std::cout<<"The Area of the Square is "<<area<<std::endl;
}

void area(float radius){
    float area;
    area=3.14*radius*radius;
    std::cout<<"The Area of the Circle is "<<area<<std::endl;
}

void area(float height,float base1,float base2){
    float area;
    area=0.5*(height*(base1+base2));
    std::cout<<"The Area of the Trapezium is "<<area<<std::endl;
}

void area(int side1,int side2,int side3){
    float area;
    float s=(side1+side2+side3)/2;
    area=std::sqrt(s*(s-side1)*(s-side2)*(s-side3));
    std::cout<<"The Area of the Triangle is "<<area<<std::endl;
}
```

```
int main() {
    int choice,option;
    do{
        std::cout<<"1.Area of Rectangle\n2.Area of Circle\n3.Area of
        Trapezium\n4.Area of Triangle\n5.Area of Square\n6.Quit"<<std::endl;
        std::cin>>choice;
        switch(choice){
                case 1:
                    int l,b;
                    std::cout<<"Enter the Length & Breadth of Rectangle\n";
                    std::cin>>l>>b;
                    area(l, b);
                    break;
                case 2:
                    float radius;
                    std::cout<<"\n\nEnter the Radius of the Circle\n";
                    std::cin>>radius;
                    area(radius);
                    break;
                case 3:
                    float height,base1,base2;
                    std::cout<<"\n\nEnter the Height and Bases of Trapezium\n";
                    std::cin>>height>>base1>>base2;
                    area(height, base1, base2);
                    break;
                case 4:
                    int side1,side2,side3;
                    std::cout<<"\n\nEnter the Sides of Triangle\n";
                    std::cin>>side1>>side2>>side3;
                    area(side1, side2, side3);
                    break;
                case 5:
                    int side;
                    std::cout<<"\n\nEnter the Side of the Square\n";
                    std::cin>>side;
                    area(side);
                    break;
                case 6:
                    break;
        }
```

```
        std::cout<<"Do you want to continue?\n1.Continue\n2.Quit\n";
        std::cin>>option;
        }
        while(option == 1);
        return 0;
    }
```

SAMPLE INPUT-OUTPUT

```
1.Area of Rectangle
2.Area of Circle
3.Area of Trapezium
4.Area of Triangle
5.Area of Square
6.Quit
1
Enter the Length & Breadth of Rectangle
2 3
The Area of the Rectangle is 6
2
Enter the Radius of the Circle
7
The Area of the Circle is 153.86
3
Enter the Height and Bases of Trapezium
5
6 7
The Area of the Trapezium is 32.5
4
Enter the Sides of Triangle
3 4 5
The Area of the Triangle is 6
5
Enter the Side of the Square
4
The Area of the Square is 16
```

BANK TRANSACTIONS

AIM

Write a C++ program using classes to perform bank transaction for n customers (cust name, acc no, acc type, balance). The program should be menu driven and it should have the following menus like adding new account, withdraw (keep a min balance of 500), deposit, balance enquiry and account statement (cust name, acc no, acc type, balance).

PROGRAM

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

class bank{
    string cust_name;
    int acc_no;
    char acc_type;
    float balance,customer_amount;
    int static count;
public:
    void new_acc(void);
    void withdraw(void);
    void deposit(void);
    void balance_enquiry(void);
    void acc_statement(void);
};
int bank::count;

void bank::new_acc(){
    count++;
    acc_no=count;
    cin.ignore(numeric_limits<streamsize>::max(),'\n');
    cout<<"Enter your name"<<endl;
    getline(cin,cust_name);
    cout<<"Enter the type of Account your prefer\nS for Savings
    Account\nC for Current Account\n";
    cin>>acc_type;
    cout<<"Enter the Amount you want to deposit (Minimum Balance is Rs.500)\n";
    cin>>balance;
    cout<<"\nYour Account Number is "<<acc_no<<endl;
```

```
    cout<<"Congratulations , Account Created Successfully\n";
}

void bank::withdraw(){
    cout<<"Enter the amount you have to withdraw\n";
    cin>>customer_amount;
    if ((balance-customer_amount)>500) {
        balance=balance-customer_amount;
        cout<<"Rs."<<customer_amount<<" is withdrawn\n";
    } else {
        cout<<"Transaction Failed\nInfo : Minimum Balance Not Available\n";
    }
}

void bank::deposit(){
    cout<<"Enter the amount you have to deposit\n";
    cin>>customer_amount;
    balance=customer_amount+balance;
    cout<<"Rs."<<customer_amount<<" is deposited\n";
}

void bank::balance_enquiry(){
    cout<<"Current Balance is Rs."<<balance<<endl;
}

void bank::acc_statement(){
    cout<<"-------------------------------------\n";
    cout<<"Your Account Statement is \n";
    cout<<"Account Holder  : "<<cust_name<<endl;
    cout<<"Account No      : "<<acc_no<<endl;
    cout<<"Account Type    : "<<acc_type<<endl;
    cout<<"Account Balance : Rs."<<balance<<endl;
    cout<<"-------------------------------------\n";
}

int main() {
    const int customer_number=20;
    int choice;
    bool Close = false;
    bank* customer=new bank[customer_number];
    int leave,exitOption;
```

```
cout<<"1.Start\n2.Quit"<<endl;
cin>>exitOption;
if (exitOption==2) {
    return 0;
}
for (int i = 0; i < customer_number; ++i) {
    if(Close){
        leave = 3;
    }
    else{
        leave = 1;
    }
    while (leave == 1){
        cout<<"\nCustomer "<<i+1<<endl;
        cout<<"Main Menu\n1.Press 1 for New Account\n2.Press 2 to
        Withdraw\n3.Press 3 to Deposit\n4.Press 4 to check balance\n
        5.Press 5 for Account Statement\n";
        cin>>choice;
        switch (choice) {
            case 1:
                customer[i].new_acc();
                break;
            case 2:
                customer[i].withdraw();
                break;
            case 3:
                customer[i].deposit();
                break;
            case 4:
                customer[i].balance_enquiry();
                break;
            case 5:
                customer[i].acc_statement();
                break;
            default:
                cout<<"Error! , Try Again"<<endl;
                break;
        }
        cout<<"\nDo you want to continue or quit.\nPress 1 to Continue
        \nPress 2 to Next Customer\nPress 3 to Quit the Application\n";
        cin>>leave;
```

```
            if (leave==3) {
                Close = true;
            }
        }
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1


Customer 1
Main Menu
1.Press 1 for New Account
2.Press 2 to Withdraw
3.Press 3 to Deposit
4.Press 4 to check balance
5.Press 5 for Account Statement
1
Enter your name
Hakkeem
Enter the type of Account your prefer
S for Savings Account
C for Current Account
S
Enter the Amount you want to deposit (Minimum Balance is Rs.500)
2500

Your Account Number is 1
Congratulations , Account Created Successfully
```

```
2
Enter the amount you have to withdraw
3000
Transaction Failed
Info : Minimum Balance Not Available
```

```
2
Enter the amount you have to withdraw
1500
Rs.1500 is withdrawn
```

```
3
Enter the amount you have to deposit
6500
Rs.6500 is deposited
```

```
4
Current Balance is Rs.7500
```

```
5
-----------------------------------
Your Account Statement is
Account Holder  : Hakkeem
Account No      : 1
Account Type    : S
Account Balance : Rs.7500
-----------------------------------
```

STRING OPEARTIONS

AIM

Write a C++ progtram to perform operations such as compare, concatenate and length on String objects.

PROGRAM

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

class Strings{
    char *name;
    int length;
public:
    Strings(){
        length=0;
        name = new char[length+1];
    }
    ~Strings(){
        delete name;
    }
    void getString(char *txt);
    void compare(const Strings&,const Strings&);
    void concatenate(const Strings&,const Strings&);
    void displayLength();
};

void Strings::getString(char *txt){
    length = strlen(txt);
    delete name;
    name = new char[length+1];
    strcpy(name, txt);
}

void Strings::concatenate(const Strings &string1,const Strings &string2){
    length = string1.length + string2.length;
    delete name;
    name = new char[length+1];
    strcpy(name, string1.name);
```

```
    strcat(name, string2.name);
    cout<<name<<endl;
}

void Strings::compare(const Strings &string1,const Strings &string2){
    if (string1.length == string2.length) {
        cout<<"They are same strings"<<endl;
    }
    else if (string1.length > string2.length) {
        cout<<string1.name<<" is greater than "<<string2.name<<endl;
    }
    else {
        cout<<string2.name<<" is greater than "<<string1.name<<endl;
    }
}

void Strings::displayLength(){
    cout<<"The String Length is "<<length<<endl;
}

int main() {
    char name1[50],name2[50];
    Strings string1,string2,result;
    int choice,option,exitOption;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==2) {
        return 0;
    }
    cout<<"Enter your First String"<<endl;
    cin>>name1;
    cout<<"Enter your Second String"<<endl;
    cin>>name2;
    string1.getString(name1);
    string2.getString(name2);
    do
    {
        cout<<"1.String Length\n2.Concatenate\n3.Compare"<<endl;
    cin>>choice;
    switch (choice){
        case 1:
```

```
                cout<<"First String"<<endl;
                string1.displayLength();
                cout<<"Second String"<<endl;
                string2.displayLength();
                break;
            case 2:
                result.concatenate(string1, string2);
                break;
            case 3:
                result.compare(string1,string2);
                break;
            default:
                cout<<"Invalid Choice"<<endl;
                break;
        }
        cout<<"Do you want to continue or quit\n1.Continue\n2.Quit"<<endl;
        cin>>option;
    } while (option == 1);
    if(option != 1){
        cout<<"Succesfully Quitted"<<endl;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
Enter your First String
Abdul
Enter your Second String
Hakkeem
1.String Length
2.Concatenate
3.Compare
1
First String
The String Length is 5
Second String
The String Length is 7
```

```
2
AbdulHakkeem
```

```
3
Hakkeem is greater than Abdul
```

CONSTRUCTORS AND DESTRUCTORS

AIM

Write a C++ program to demonstrate the order of execution of constructors & destructors.

PROGRAM

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

class A{
    int **p;
    int d1,d2;
public:
    A(int x,int y){
        cout<<"Array dynamically allocated by constructor"<<endl;
        d1 = x;
        d2 = y;
        p = new int *[d1];
        for (int i = 0; i < d1; ++i) {
            p[i] = new int [d2];
        }
    }
    void get_data();
    void put_data();
    ~A(){
        cout<<"Memory released using destructor"<<endl;
    }
};

void A::get_data(){
    cout<<"Enter the Values Row by Row"<<endl;
    for (int i = 0; i < d1; ++i) {
        cout<<"Row - "<<i+1<<endl;
        for (int j = 0; j < d2; ++j) {
            cin>>p[i][j];
        }
    }
}

void A::put_data(){
```

```
    for (int i = 0; i < d1; ++i) {
        for (int j = 0; j < d2; ++j) {
            cout<<p[i][j]<<"\t";
        }
        cout<<endl;
    }
}

int main() {
    int row,col;
    cout<<"Enter the rows and coloumns of the matrix"<<endl;
    cin>>row>>col;
    A Object(row,col);
    Object.get_data();
    Object.put_data();
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the rows and coloumns of the matrix
2 2
Array dynamically allocated by constructor
Enter the Values Row by Row
Row - 1
1 2
Row - 2
3 4
1       2
3       4
Memory released using destructor
```

TIME CLASS

AIM

Create a class TIME with members hours, minutes, seconds. Take input, add two time objects by passing objects to function and display result.

PROGRAM

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

class Time{
    int hours,minutes,seconds;
public:
    void gettime(){
        cout<<"Enter the Hour\n";
        cin>>hours;
        cout<<"Enter the Minute\n";
        cin>>minutes;
        cout<<"Enter the Seconds\n";
        cin>>seconds;
    }
    void puttime(void){
        cout<<hours<<" hours "<<minutes<<" minutes and "<<seconds<<" seconds"<<endl;
    }
    void sum(Time,Time);
};

void Time::sum(Time t1,Time t2){
    seconds = t1.seconds+t2.seconds;
    minutes = seconds/60;
    seconds = seconds%60;
    minutes = minutes+t1.minutes+t2.minutes;
    hours=minutes/60;
    minutes=minutes%60;
    hours=hours+t1.hours+t2.hours;
}

int main() {
    Time T1,T2,T3;
    cout<<"Time 1"<<endl;
```

```
    T1.gettime();
    cout<<"Time 2"<<endl;
    T2.gettime();
    T3.sum(T1, T2);
    cout<<"Time 1 = ";
    T1.puttime();
    cout<<"Time 2 = ";
    T2.puttime();
    cout<<"Result  = ";
    T3.puttime();
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Time 1
Enter the Hour
2
Enter the Minute
3
Enter the Seconds
45
Time 2
Enter the Hour
4
Enter the Minute
45
Enter the Seconds
15
Time 1 = 2 hours 3 minutes and 45 seconds
Time 2 = 4 hours 45 minutes and 15 seconds
Result  = 6 hours 49 minutes and 0 seconds
```

MATRIX CLASS AND ITS OPERATIONS

AIM

Write a C++ program to implement a class MATRIX with member functions such as matrix_add, matrix_mult, matrix_transpose, matrix_determinant etc.

PROGRAM

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

class matrices{
    int rows,coloumns;
    int **matrix;
public:
    matrices(int r,int c){
        rows = r;
        coloumns = c;
        matrix = new int *[rows];
        for (int i = 0; i < rows; ++i) {
            matrix[i]= new int [coloumns];
        }
    }
    ~matrices(){
        for (int i = 0; i < rows; ++i) {
            delete matrix[i];
        }
        delete matrix;
        cout<<"Memory Released"<<endl;
    }
    matrices(){};
    void get_matrix();
    friend void matrix_add(const matrices&,const matrices&);
    friend void matrix_mult(const matrices&,const matrices&);
    void matrix_transpose();
    void matrix_trace();
};

void matrices::get_matrix(){
    for (int i = 0; i < rows; ++i) {
        cout<<"Enter elements of "<<i+1<<" row"<<endl;
```

```
        for (int j = 0; j < coloumns; ++j) {
            cin>>matrix[i][j];
        }
    }
}
void matrix_add(const matrices &a,const matrices &b){
    if(a.rows==b.rows and a.coloumns==b.coloumns){
        int sum[a.rows][a.coloumns];
        for (int i = 0; i < a.rows; ++i) {
            for (int j = 0; j < a.coloumns; ++j) {
                sum[i][j]=a.matrix[i][j]+b.matrix[i][j];
            }
        }
        for (int i = 0; i < a.rows; ++i) {
            for (int j = 0; j < b.coloumns; ++j) {
                cout<<sum[i][j]<<"\t";
            }
            cout<<endl;
        }
    }
    else{
        cout<<"Addition not possible"<<endl;
    }
}

void matrices::matrix_transpose(){
    int transpose[coloumns][rows];
    //to take the transpose
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < coloumns; ++j) {
            transpose[j][i] = matrix[i][j];
        }
    }
    //to display the transpose
    cout<<"Transpose of the Matrix"<<endl;
    for (int i = 0; i < coloumns; ++i) {
        for (int j = 0; j < rows; ++j) {
            cout<<transpose[i][j]<<"\t";
        }
        cout<<"\n";
    }
```

```
}

void matrix_mult(const matrices &a,const matrices &b){
    if (a.coloumns==b.rows) {
        int mult[a.rows][b.coloumns];
        for (int i = 0; i < a.rows; ++i) {
            for (int j = 0; j < b.coloumns; ++j) {
                int sum = 0;
                for (int k = 0; k < a.coloumns; ++k) {
                    sum = sum + a.matrix[i][k]*b.matrix[k][j];
                    mult[i][j]=sum;
                }
            }
        }
        //printing result
        cout<<"The Result is"<<endl;
        for (int i = 0; i < a.rows; ++i) {
            for (int j = 0; j < b.coloumns; ++j) {
                cout<<mult[i][j]<<"\t";
            }
            cout<<endl;
        }
    } else {
        cout<<"For Matrix Multiplication , no of coloumns of first
        matrix should be equal to no of rows of second matrix"<<endl;
    }
}

void matrices::matrix_trace(){
    if (rows !=coloumns){
        cout<<"Trace is possible for square matrix"<<endl;
    }
    else{
        int trace = 0;
        for (int i = 0; i < rows; ++i) {
            trace = trace + matrix[i][i];
        }
        cout<<"The Trace of the Matrix is "<<trace<<endl;
    }
}
```

```
int main() {
    int Exit;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>Exit;
    if (Exit==2) {
        return 0;
    }
    int choice,m,n,opt,exit_option;
    cout<<"Enter the size of the First Matrix"<<endl;
    cin>>m>>n;
    matrices matrix1(m,n);
    matrix1.get_matrix();
    cout<<"Enter the size of the Second Matrix"<<endl;
    cin>>m>>n;
    matrices matrix2(m,n);
    matrix2.get_matrix();
    do {
        cout<<"Main Menu"<<endl;
        cout<<"Welcome , select the operation you want to perform"<<endl;
        cout<<"1.Addition of Matrix\n2.Transpose of Matrix\n3.Matrix
        Multiplication\n4.Trace of Matrix"<<endl;
        cin>>choice;
        switch (choice) {
            case 1:
                matrix_add(matrix1, matrix2);
                break;
            case 2:
                cout<<"Choose the option\n1.Transpose of First Matrix\n2.
                Transpose of Second Matrix"<<endl;
                cin>>opt;
                if(opt == 1){
                    matrix1.matrix_transpose();
                }
                else {
                    matrix2.matrix_transpose();
                }
                break;
            case 3:
                matrix_mult(matrix1, matrix2);
                break;
            case 4:
```

```
                cout<<"Choose the option\n1.Trace of First Matrix\n2.Trace of
                Second Matrix"<<endl;
                cin>>opt;
                if(opt == 1){
                    matrix1.matrix_trace();
                }
                else {
                    matrix2.matrix_trace();
                }
                break;
            default:
                cout<<"Invalid Choice";
                break;
        }
        cout<<"Do you want to continue?\n1.Continue\n2.Exit"<<endl;
        cin>>exit_option;
    } while (exit_option==1);
    if (exit_option!=1){
        cout<<"Succesfully Exited"<<endl;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
Enter the size of the First Matrix
2 2
Enter elements of 1 row
1 2
Enter elements of 2 row
3 4
Enter the size of the Second Matrix
2 2
Enter elements of 1 row
3 4
Enter elements of 2 row
5 6
```

```
Main Menu
Welcome , select the operation you want to perform
1.Addition of Matrix
2.Transpose of Matrix
3.Matrix Multiplication
4.Trace of Matrix
1
4       6
8       10
2
Choose the option
1.Transpose of First Matrix
2.Transpose of Second Matrix
1
Transpose of the Matrix
1       3
2       4
2
Transpose of the Matrix
3       5
4       6
3
The Result is
13      16
29      36
4
Choose the option
1.Trace of First Matrix
2.Trace of Second Matrix
1
The Trace of the Matrix is 5
2
The Trace of the Matrix is 9
```

```
Enter the size of the First Matrix
1 3
Enter elements of 1 row
1 2 3
Enter the size of the Second Matrix
2 3
Enter elements of 1 row
1 2 3
Enter elements of 2 row
4 5 6
Main Menu
Welcome , select the operation you want to perform
1.Addition of Matrix
2.Transpose of Matrix
3.Matrix Multiplication
4.Trace of Matrix
1
Addition not possible
3
For Matrix Multiplication , no of coloumns of first matrix should
be equal to no of rows of second matrix
```

CONSTRUCTOR OVERLOADING

AIM

Write a program to perform addition of two complex numbers using constructor overloading. The first constructor which takes no argument is used to create objects which are not initialized, second which takes one argument is used to initialize real and imag parts to equal values and third which takes two argument is used to initialized real and imag to two different values.

PROGRAM

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

class complex{
    float real,image;
public:
    complex(){}
    complex(float a){
        real=image=a;
    }
    complex(float x,float y){
        real=x;
        image=y;
    }
    friend complex sum(complex,complex);
    friend void display(complex);
};

complex sum(complex A,complex B){
    complex result;
    result.real=A.real+B.real;
    result.image=A.image+B.image;
    return result;
}

void display(complex number){
    if (number.image<0) {
        cout<<number.real<<" "<<number.image<<"i"<<endl;
    } else {
        cout<<number.real<<" + "<<number.image<<"i"<<endl;
```

```
    }
}

int main() {
    int exitOption,loopOption;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==2) {
        return 0;
    }
    do {
        float num1,num2;
        cout<<"Enter the different real and image part\n";
        cin>>num1>>num2;
        complex A(num1,num2);
        cout<<"A = ";
        display(A);
        float num3;
        cout<<"Enter the same real and image part"<<endl;
        cin>>num3;
        complex B(num3);
        cout<<"B = ";
        display(B);
        complex C;
        C=sum(A,B);
        cout<<"Sum = ";
        display(C);
        cout<<"Do you want to continue ?\n1.Continue\n2.Quit"<<endl;
        cin>>loopOption;
    } while (loopOption==1);
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
Enter the the different real and image part
2 3
A = 2 + 3i
Enter the same real and image part
2
B = 2 + 2i
Sum = 4 + 5i
```

STATIC MEMBER FUNCTION

AIM

Write a C++ program to design a class having static member function named showcount() which has the property of displaying the number of objects created of the class.

PROGRAM

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

class object{
    int static count;
public:
    object(){
        count++;
    }
    void static showcount(){
        cout<<"No of Objects created : "<<count<<endl;
    }
};

int object::count;

int main() {
    int choice,exitOption;
    cout<<"1.Create Object\n2.Quit"<<endl;
    cin>>exitOption;
    if(exitOption==2){
        return 0;
    }
    do {
        object *a = new object;
        a->showcount();
        delete a;
        cout<<"Do you want to create new objects \n1.New Object \n2.Quit"<<endl;
        cin>>choice;
    } while (choice==1);
    return 0;

}
```

SAMPLE INPUT-OUTPUT

```
1.Create Object
2.Quit
1
No of Objects created : 1
Do you want to create new objects
1.New Object
2.Quit
1
No of Objects created : 2
```

DEPARTMENTAL STORE

AIM

Write a C++ program using class to process shopping list for a Departmental Store. The list include details such as the Code-no and price of each item and perform the operations like adding deleting items to the list and printing the total value of an order.

PROGRAM

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

class list{
    string itemName;
    int itemCode;
    float itemPrice;
    int itemQuantity;
    float totalSum;
public:
    list(){
        itemCode=0;
        itemPrice=0;
        itemQuantity=0;
        totalSum = 0;
    }
    void addItem(){
        cout<<"Item Name : ";
        cin>>itemName;
        cout<<"Item Code : ";
        cin>>itemCode;
        cout<<"Item Quantity : ";
        cin>>itemQuantity;
        cout<<"Item Price : ";
        cin>>itemPrice;
    }
    void displayItem(){
        cout<<itemCode<<"\t\t"<<itemName<<"\t\t"<<itemPrice<<"\t\t"
        <<itemQuantity<<endl;
    }
    void alterQuantity(int decrement){
        itemQuantity = itemQuantity - decrement;
```

```
    }
    int checkCode(){
        return itemCode;
    }
    int checkQuantity(){
        return itemQuantity;
    }
    void updateQuantity(int newQuantity){
        itemQuantity = itemQuantity + newQuantity;
    }
    float generateSum(){
        return itemQuantity*itemPrice;
    }
    void newQuantity(int newQuantity){
        itemQuantity = newQuantity;
    }
};

int main() {
    list *stock = new list[10];
    list *shoppingList = new list[10];
    int stockCount = 0;
    int shoppingCount = 0;
    int option,choice;
    int keyPosition;
    bool found = false;
    do {
        cout<<"1.Add Item to Stock\n2.Delete Item\n3.Update Item
        \n4.Display Items\n5.Purchase Item"<<endl;
        cin>>option;
        switch (option) {
            case 1:
                stock[stockCount].addItem();
                cout<<"Item Added Successfully"<<endl;
                stockCount++;
                break;
            case 2:
            {
                int searchKey;
                cout<<"Item Code : ";
                cin>>searchKey;
```

```
        for (int i = 0; i < stockCount; ++i) {
            if (searchKey==stock[i].checkCode()) {
                found = true;
                keyPosition = i;
                cout<<"Successfully Deleted"<<endl;
                break;
            }
        }
        if (!found) {
            cout<<"Invalid Code"<<endl;
        }
        else{
            for (int i = keyPosition; i < stockCount; ++i){
                stock[i]=stock[i+1];
            }
            stockCount--;
        }
    }
        break;
    case 3:
    {
        bool found = false;
        cout<<"Item Code\tItem Name\tItem Price\tItem
        Quantity"<<endl;
        for (int i = 0; i < stockCount; ++i) {
            stock[i].displayItem();
        }
        int searchKey;
        cout<<"Enter the Item Code : ";
        cin>>searchKey;
        for (int i = 0; i < stockCount; ++i) {
            if (searchKey==stock[i].checkCode()) {
                int x;
                cout<<"Enter the Amount of Quantity : ";
                cin>>x;
                stock[i].updateQuantity(x);
                found = true;
                cout<<"Successfully Updated"<<endl;
                break;
            }
        }
```

```
        if(!found){
            cout<<"Invalid Code"<<endl;
        }
    }
        break;
    case 4:
        cout<<"Item Code\tItem Name\tItem Price\tItem
        Quantity"<<endl;
        for (int i = 0; i < stockCount; ++i) {
            stock[i].displayItem();
        }
        break;
    case 5:
        int choice;
        do{
            bool purchased = false;
            int tempItemCode,tempItemQuantity;
            cout<<"Enter the Item Code"<<endl;
            cin>>tempItemCode;
            cout<<"Quantity you want to purchase"<<endl;
            cin>>tempItemQuantity;
            for (int i = 0; i < stockCount; ++i) {
                if (tempItemCode==stock[i].checkCode()) {
                    if(tempItemQuantity>stock[i].checkQuantity()){
                        cout<<"Insufficient Stock"<<endl;
                        break;
                    }
                    purchased = true;
                    shoppingList[shoppingCount]=stock[i];
                    shoppingList[shoppingCount].newQuantity
                    (tempItemQuantity);
                    stock[i].alterQuantity(tempItemQuantity);
                    shoppingList[shoppingCount].displayItem();
                    shoppingCount++;
                    break;
                }
            }
            cout<<"Do you want to continue purchase"<<endl;
            cout<<"1.Continue\n2.Generate Bill"<<endl;
            cin>>choice;
            float totalSum;
```

```
                    if(choice==2 and purchased==true){
                        cout<<"Your Purchase Bill"<<endl;
                        cout<<"Item Code\tItem Name\tItem
                        Price\tItem Quantity"<<endl;
                        for (int i = 0; i < shoppingCount; ++i) {
                            shoppingList[i].displayItem();
                            totalSum = totalSum +
                            shoppingList[i].generateSum();
                        }
                    cout<<"Total Sum = "<<totalSum<<endl;
                    }
                } while(choice==1);
                break;
            default:
                break;
        }
        cout<<"1.Continue\n2.Quit"<<endl;
        cin>>choice;
    } while(choice==1);
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Add Item
2.Delete Item
3.Update Item
4.Display Items
5.Purchase Item
1
Item Name : Milk
Item Code : 1
Item Quantity : 1000
Item Price : 50
Item Name : Soap
Item Code : 2
Item Quantity : 2500
Item Price : 100
Item Name : Bread
Item Code : 3
Item Quantity : 5000
Item Price : 150
```

```
2
Item Code : 4
Invalid Code
```

```
2
Item Code : 3
Successfully Deleted
```

```
3
Item Code       Item Name       Item Price      Item Quantity
1               Milk            50              1000
2               Soap            100             2500
Enter the Item Code : 1
Enter the Amount of Quantity : 2000
Successfully Updated
```

```
4
Item Code       Item Name       Item Price      Item Quantity
1               Milk            50              3000
2               Soap            100             2500
```

```
5
Enter the Item Code
1
Quantity you want to purchase
4000
Insufficient Stock
```

```
Enter the Item Code
1
Quantity you want to purchase
2000
```

```
Your Purchase Bill
Item Code       Item Name       Item Price      Item Quantity
1               Milk            50              2000
Total Sum = 100000
```

```
Item Code       Item Name       Item Price      Item Quantity
1               Milk            50              1000
2               Soap            100             2500
```

SWAPPING PRIVATE DATA MEMBERS

AIM

Write a Program to swap private data members of classes named as class_1, class_2 using friend function.

PROGRAM

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

class class_2;
class class_1{
    int num1;
public:
    class_1(){};
    void get_value_firstClass(){
        cout<<"Enter the value for Private Data Member of Class 1"<<endl;
        cin>>num1;
    }
    friend void swap(class_1&,class_2&);
    void display_class1(){
        cout<<"Class 1 = "<<num1<<endl;
    }
};

class class_2{
    int num2;
public:
    class_2(){};
    void get_value_secClass(){
        cout<<"Enter the value for Private Data Member of Class 2"<<endl;
        cin>>num2;
    }
    friend void swap(class_1&,class_2&);
    void display_class2(){
        cout<<"Class 2 = "<<num2<<endl;
    }
};

void swap(class_1 &a,class_2 &b){
```

```
    int temp;
    temp=a.num1;
    a.num1=b.num2;
    b.num2=temp;
}

int main() {
    int exitOption;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==2) {
        return 0;
    }
    class_1 obj1;
    class_2 obj2;
    obj1.get_value_firstClass();
    obj2.get_value_secClass();
    cout<<"Before Swapping"<<endl;
    obj1.display_class1();
    obj2.display_class2();
    swap(obj1, obj2);
    cout<<"\nAfter Swapping"<<endl;
    obj1.display_class1();
    obj2.display_class2();
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
Enter the value for Private Data Member of Class 1
2
Enter the value for Private Data Member of Class 2
4
Before Swapping
Class 1 = 2
Class 2 = 4

After Swapping
Class 1 = 4
Class 2 = 2
```

COMPLEX CLASS

AIM

Write a Program to design a class complex to represent complex numbers. The complex class should use an external function (use it as a friend function) to add two complex numbers. The function should return an object of type complex representing the sum of two complex numbers.

PROGRAM

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

class complex{
    int real,image;
public:
    void getvalue();
    complex friend sum(complex,complex);
    void display();
};
void complex::getvalue(){
    cout<<"Enter the Real Part ";
    cin>>real;
    cout<<"Enter the Image Part ";
    cin>>image;
}

void complex::display(){
    if (image<0) {
        cout<<real<<image<<"i"<<endl;
    } else {
        cout<<real<<"+"<<image<<"i"<<endl;
    }
}

complex sum(complex a,complex b){
    complex result;
    result.real=a.real+b.real;
    result.image=a.image+b.image;
    return result;
}
```

```
int main() {
    int exitOption;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==2) {
        return 0;
    }
    complex a,b,result;
    cout<<"Complex 1"<<endl;
    a.getvalue();
    cout<<"Complex 2"<<endl;
    b.getvalue();
    cout<<"\nComplex 1 : ";
    a.display();
    cout<<"Complex 2 : ";
    b.display();
    result=sum(a, b);
    cout<<"Result    : ";
    result.display();
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
Complex 1
Enter the Real Part 2
Enter the Image Part 3
Complex 2
Enter the Real Part 4
Enter the Image Part -5

Complex 1 : 2+3i
Complex 2 : 4-5i
Result    : 6-2i
```

OPERATOR OVERLOADING FOR VECTOR CLASS

AIM

Write a C++ program to overload ==, ! =, <, <=, > and >= operators as member operator functions for a vector object.

PROGRAM

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

class Vector{
    float i_Component,j_Component,k_Component,magnitude;
public:
    void getVector();
    void displayVector();
    void FindMagnitude();
    void operator==(const Vector&);
    void operator!=(const Vector&);
    void operator<(const Vector&);
    void operator<=(const Vector&);
    void operator>(const Vector&);
    void operator>=(const Vector&);
};

//member function to get the components of the vector.
void Vector::getVector(){
    cout<<"Enter the component of i"<<endl;
    cin>>i_Component;
    cout<<"Enter the component of j"<<endl;
    cin>>j_Component;
    cout<<"Enter the component of k"<<endl;
    cin>>k_Component;
}

//member function to display the vector entered.
void Vector::displayVector(){
    if (j_Component>=0 and k_Component>=0) {
        cout<<i_Component<<"i + "<<j_Component<<"j + "<<k_Component<<"k"<<endl;
}
```

```
else {
        if(j_Component<0 and k_Component<0){
            cout<<i_Component<<"i "<<j_Component<<"j "<<k_Component<<"k"<<endl;
}
        else if(j_Component<0){
            cout<<i_Component<<"i "<<j_Component<<"j + "<<k_Component<<"k"<<endl;
        }
        else {
            cout<<i_Component<<"i + "<<j_Component<<"j "<<k_Component<<"k"<<endl;
        }
    }
}

//member function to compute the magnitude of the vector
void Vector::FindMagnitude(){
    magnitude = (i_Component*i_Component)+(j_Component*j_Component)+
    (k_Component*k_Component);
    magnitude = sqrtf(magnitude);
}

//member function to overload == operator for vector
void Vector::operator==(const Vector &A){
    if (magnitude == A.magnitude){
        cout<<"They are equal vectors"<<endl;
    }
    else {
        cout<<"They are unequal vectors"<<endl;
    }
}
//member function to overload != operator for vector
void Vector::operator!=(const Vector &A){
    if (magnitude != A.magnitude){
        cout<<"They are unequal vectors"<<endl;
    }
    else {
        cout<<"They are equal vectors"<<endl;
    }
}
//member function to overload < operator for vector
void Vector::operator<(const Vector &A){
    if (magnitude < A.magnitude) {
```

```
        cout<<"Vector 2 is greater than Vector 1"<<endl;
    }
    else {
        cout<<"Vector 1 is greater than Vector 2"<<endl;
    }
}
//member function to overload <= operator for vector
void Vector::operator<=(const Vector &A){
    if (magnitude < A.magnitude) {
        cout<<"Vector 2 is greater than Vector 1"<<endl;
    }
    else if(magnitude == A.magnitude){
        cout<<"They are equal vectors"<<endl;
    }
    else {
        cout<<"Vector 1 is greater than Vector 2"<<endl;
    }
}
//member function to overload > operator for vector
void Vector::operator>(const Vector &A){
    if (magnitude > A.magnitude) {
        cout<<"Vector 1 is greater than Vector 2"<<endl;
    }
    else {
        cout<<"Vector 2 is greater than Vector 1"<<endl;
    }
}
//member function to overload >= operator for vector
void Vector::operator>=(const Vector &A){
    if (magnitude > A.magnitude) {
        cout<<"Vector 1 is greater than Vector 2"<<endl;
    }
    else if(magnitude == A.magnitude){
        cout<<"They are equal vectors"<<endl;
    }
    else {
        cout<<"Vector 2 is greater than Vector 1"<<endl;
    }
}

int main() {
```

```
int option;
cout<<"Welcome\n1.Input Vectors\n2.Quit"<<endl;
cin>>option;
if (option==2) {
    cout<<"You have successfully quited"<<endl;
    return 0;
}
Vector V1,V2;
int choice,choice2;
cout<<"Vector 1"<<endl;
V1.getVector();
cout<<"\nVector 2"<<endl;
V2.getVector();
cout<<"Vector 1 = ";
V1.displayVector();
cout<<"Vector 2 = ";
V2.displayVector();
V1.FindMagnitude();
V2.FindMagnitude();

do {
    cout<<"\nChoose the operation you want to perform on the vector \n1.
    Vector1 == Vector2\n2.Vector1 != Vector2\n3.Vector1 <  Vector2
    \n4.Vector1 <= Vector2\n5.Vector1 >  Vector2\n6.Vector1 >= Vector2
    \n7.Quit"<<endl;
    cin>>choice;
    switch (choice) {
        case 1:
            V1== V2;
            break;
        case 2:
            V1!= V2;
            break;
        case 3:
            V1< V2;
            break;
        case 4:
            V1<= V2;
            break;
        case 5:
            V1> V2;
```

```
                break;
            case 6:
                V1>= V2;
                break;
            default:
                break;
        }
        cout<<"Do you want to Continue ?\n1.Continue\n2.Quit"<<endl;
        cin>>choice2;
    } while (choice2 == 1);
    if(choice2!=1){
        cout<<"Successfully Quitted!"<<endl;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Welcome
1.Input Vectors
2.Quit
1
Vector 1
Enter the component of i
2
Enter the component of j
3
Enter the component of k
5

Vector 2
Enter the component of i
1
Enter the component of j
2
Enter the component of k
3
Vector 1 = 2i + 3j + 5k
Vector 2 = 1i + 2j + 3k
Choose the operation you want to perform on the vector
1.Vector1 == Vector2
2.Vector1 != Vector2
3.Vector1 <  Vector2
4.Vector1 <= Vector2
5.Vector1 >  Vector2
6.Vector1 >= Vector2
7.Quit
1
They are unequal vectors
```

```
2
They are unequal vectors
```

```
3
Vector 1 is greater than Vector 2
```

```
4
Vector 1 is greater than Vector 2
```

```
5
Vector 1 is greater than Vector 2
```

```
6
Vector 1 is greater than Vector 2
```

OPERATOR OVERLOADING FOR COMPLEX CLASS

AIM

Write a C++ program to design a class representing complex numbers and having the functionality of performing addition & multiplication of two complex numbers using operator overloading (Use friend operator functions).

PROGRAM

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

class Complex{
    float real,image;
public:
    void getComplex();
    void displayComplex();
    friend Complex operator+(const Complex&,const Complex&);
    friend Complex operator*(const Complex&,const Complex&);
};
//member function to get the complex number.
void Complex::getComplex(){
    cout<<"Enter the real part"<<endl;
    cin>>real;
    cout<<"Enter the image part"<<endl;
    cin>>image;
}
//member function to display the complex number.
void Complex::displayComplex(){
    if (image<0) {
        cout<<real<<" "<<image<<"i"<<endl;
    } else {
        cout<<real<<" + "<<image<<"i"<<endl;
    }
}
//friend function to overload + for complex addition
Complex operator+(const Complex &A,const Complex &B){
    Complex Sum;
    Sum.real = A.real+B.real;
    Sum.image = A.image+B.image;
    return Sum;
```

```
}
//friend function to overload * for complex multiplication
Complex operator*(const Complex &A,const Complex &B){
    Complex Product;
    Product.real = (A.real*B.real)-(A.image*B.image);
    Product.image = (A.image*B.real) + (A.real*B.image);
    return Product;
}
int main() {
    int quit;
    cout<<"Welcome\n1.Start\n2.Quit"<<endl;
    cin>>quit;
    if (quit==2) {
        cout<<"You have successfully quitted"<<endl;
        return 0;
    }
    int choice,choice2;
    Complex complex1,complex2,Sum,Product;
    cout<<"Complex Number - 1"<<endl;
    complex1.getComplex();
    cout<<"Complex Number - 2"<<endl;
    complex2.getComplex();
    cout<<"Complex Number 1 = ";
    complex1.displayComplex();
    cout<<"Complex Number 2 = ";
    complex2.displayComplex();
    do {
        cout<<"Choose the operation you want to perform\n1.Complex
        Addition\n2.Complex Multiplication"<<endl;
        cin>>choice;
        switch (choice) {
            case 1:
                Sum = complex1+complex2;
                cout<<"Sum = ";
                Sum.displayComplex();
                break;
            case 2:
                Product = complex1*complex2;
                cout<<"Product = ";
                Product.displayComplex();
                break;
```

```
            default:
                break;
        }
        cout<<"Do you want to continue ? \n1.Continue\n2.Quit"<<endl;
        cin>>choice2;
    } while (choice2 == 1);
    if(choice2!=1){
        cout<<"Successfully Quitted!"<<endl;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Welcome
1.Start
2.Quit
1
Complex Number - 1
Enter the real part
2
Enter the image part
3
Complex Number - 2
Enter the real part
4
Enter the image part
5
Complex Number 1 = 2 + 3i
Complex Number 2 = 4 + 5i
Choose the operation you want to perform
1.Complex Addition
2.Complex Multiplication
1
Sum = 6 + 8i
2
Product = -7 + 22i
```

OPERATOR OVERLOADING FOR VECTOR CLASS

AIM

Write a C++ program to overload operators like *, <<, >> using friend function. The following overloaded operators should work for a class vector.

PROGRAM

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

class Vector{
    float iComponent,jComponent,kComponent;
public:
    float operator*(const Vector&);
    friend ostream & operator<<(ostream&,Vector&);
    friend istream & operator>>(istream&,Vector&);
};

//overloading >> operator to input the vector.
istream & operator>>(istream&din,Vector&a){
    cout<<"Enter the component of i"<<endl;
    cin>>a.iComponent;
    cout<<"Enter the component of j"<<endl;
    cin>>a.jComponent;
    cout<<"Enter the component of k"<<endl;
    cin>>a.kComponent;
    return (din);
}

//overloading << operator to input the vector.
ostream & operator<<(ostream&dout,Vector&a){
    if (a.jComponent>0 and a.kComponent>0) {
        dout<<a.iComponent<<"i + "<<a.jComponent<<"j + "<<a.kComponent<<"k"<<endl;
    }
    else {
        if(a.jComponent<0 and a.kComponent<0){
            dout<<a.iComponent<<"i "<<a.jComponent<<"j "<<a.kComponent<<"k"<<endl;
        }
        else if(a.jComponent<0){
            dout<<a.iComponent<<"i "<<a.jComponent<<"j + "<<a.kComponent<<"k"<<
```

```
            endl;
        }
        else {
            dout<<a.iComponent<<"i + "<<a.jComponent<<"j "<<a.kComponent
            <<"k"<<endl;
        }
    }
    return dout;
}

//overloading * operator to find the dot product.
float Vector::operator*(const Vector&a){
    float dotProduct;
    dotProduct = (iComponent*a.iComponent)+(jComponent*a.jComponent)
    +(kComponent*a.kComponent);
    return dotProduct;
}

int main() {
    int exitOption,loopOption;
    cout<<"Welcome \n1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==1) {
        do {
            cout<<"Vector - 1"<<endl;
            Vector vector1;
            cin>>vector1;
            cout<<"Vector - 2"<<endl;
            Vector vector2;
            cin>>vector2;
            cout<<"Vector 1 : "<<vector1;
            cout<<"Vector 2 : "<<vector2;
            int choice;
            cout<<"1.Dot Product of Vector 1 and Vector 2\n2.Quit"<<endl;
            cin>>choice;
            if(choice==1){
                float dotProduct;
                dotProduct = vector1*vector2;
                cout<<"Dot Product : "<<dotProduct<<endl;
                cout<<"Do you want to continue ?\n1.Continue\n2.Quit"<<endl;
                cin>>loopOption;
```

```
            } else{
                return 0;
            }
        } while (loopOption==1);
    }
    else {
        return 0;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Welcome
1.Start
2.Quit
1
Vector - 1
Enter the component of i
1
Enter the component of j
2
Enter the component of k
3
Vector - 2
Enter the component of i
4
Enter the component of j
5
Enter the component of k
6
Vector 1 : 1i + 2j + 3k
Vector 2 : 4i + 5j + 6k
1.Dot Product of Vector 1 and Vector 2
2.Quit
1
Dot Product : 32
```

OPERATOR OVERLOADING FOR MATRIX CLASS

AIM

Write a C++ program for developing a matrix class which can handle integer matrices of different dimensions. Also overload the operator for addition and multiplication of matrices. Use double pointers in your program to dynamically allocate memory for the matrices.

PROGRAM

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

class Matrix{
    int rows,columns;
    int **matrix;
public:
    Matrix(){}
    Matrix(int r,int c){
        rows=r;
        columns=c;
        matrix = new int *[rows];
        for (int i = 0; i < rows; ++i) {
            matrix[i] = new int [columns];
        }
    }
    //function to get the no of rows and columns to dynamically allocate memory
    void getRowsandColoumns(){
    cout<<"Enter the no of rows and columns"<<endl;
    cin>>rows>>columns;
    }
    void allocateMatrix();
    void getMatrix();
    void displayMatrix();
    Matrix operator+(const Matrix&);
    Matrix operator*(const Matrix&);
    //destructor used to delete the dynamically allocated memory
    ~Matrix(){
        for (int i = 0; i < rows; ++i) {
            delete matrix[i];
        }
        cout<<"Memory Deleted"<<endl;
```

```
        delete matrix;
    }
};

//function used to allocated dynamic memory to the matrix
void Matrix::allocateMatrix(){
    matrix = new int *[rows];
    for (int i = 0; i < rows; ++i) {
        matrix[i] = new int [columns];
    }
}

//function to get values to the matrix.
void Matrix::getMatrix(){
    cout<<"Enter the Values"<<endl;
    for (int i = 0; i < rows; ++i) {
        cout<<"Row - "<<i+1<<endl;
        for (int j = 0; j < columns; ++j) {
            cin>>matrix[i][j];
        }
    }
}
//function to display the matrix entered.
void Matrix::displayMatrix(){
    cout<<"The Matrix is of dimension "<<rows<<"x"<<columns<<endl;
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < columns; ++j) {
            cout<<matrix[i][j]<<"\t";
        }
        cout<<"\n";
    }
}
//overloading + operator for matrix addition
Matrix Matrix::operator+(const Matrix&a){
    if(rows==a.rows and columns==a.columns){
        Matrix sum(rows,columns);
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < columns; ++j) {
                sum.matrix[i][j]=matrix[i][j]+a.matrix[i][j];
            }
        }
```

```
        return sum;
    }
    else{
        cout<<"Addition not possible"<<endl;
    }
}
//overloading * operator for matrix addition
Matrix Matrix::operator *(const Matrix&a){
    if (columns==a.rows) {
        Matrix productMatrix(rows,a.columns);
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < a.columns; ++j) {
                int sum=0;
                for (int k = 0; k < columns; ++k) {
                    sum = matrix[i][k]*a.matrix[k][j]+sum;
                    productMatrix.matrix[i][j] = sum;
                }
            }
        }
        return productMatrix;
    } else {
        cout<<"Multiplication not possible"<<endl;
    }
}

int main() {
    int quitOption,exitLoop;
    cout<<"Welcome\n1.Start\n2.Quit"<<endl;
    cin>>quitOption;
    if (quitOption==1) {
        Matrix matrix1;
        cout<<"Matrix - 1"<<endl;
        matrix1.getRowsandColoumns();
        matrix1.allocateMatrix();
        matrix1.getMatrix();

        Matrix matrix2;
        cout<<"Matrix - 2"<<endl;
        matrix2.getRowsandColoumns();
        matrix2.allocateMatrix();
        matrix2.getMatrix();
```

```
        cout<<endl;
        cout<<"Matrix - 1"<<endl;
        matrix1.displayMatrix();
        cout<<endl;
        cout<<"Matrix - 2"<<endl;
        matrix2.displayMatrix();

        do {
            int option;
            cout<<"\n1.Matrix 1 + Matrix2\n2.Matrix 1 x Matrix2"<<endl;
            cin>>option;
            switch (option) {
                case 1:
                    Matrix Sum;
                    Sum = matrix1+matrix2;
                    Sum.displayMatrix();
                    break;
                case 2:
                    Matrix Product;
                    Product = matrix1*matrix2;
                    Product.displayMatrix();
                    break;
                default:
                    break;
            }
            cout<<"\n1.Continue\n2.Quit"<<endl;
            cin>>exitLoop;
        } while (exitLoop==1);
    } else {
        cout<<"Successfully Quitted"<<endl;
        return 0;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

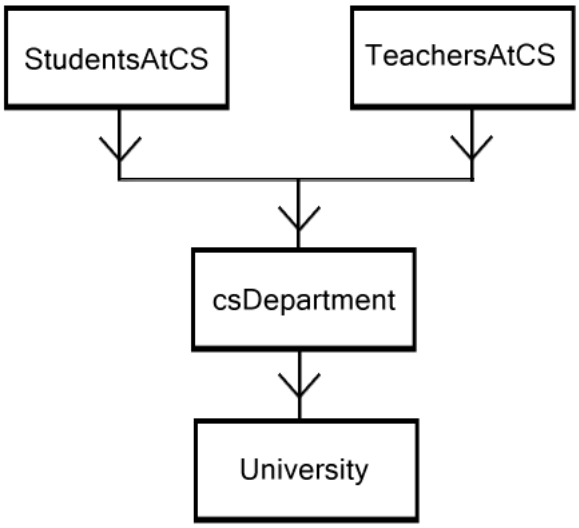
```
Welcome
1.Start
2.Quit
1
Matrix - 1
Enter the no of rows and coloumns
2 2
Enter the Values
Row - 1
1 2
Row - 2
3 4
Matrix - 2
Enter the no of rows and coloumns
2 2
Enter the Values
Row - 1
4 2
Row - 2
3 1
Matrix - 1
The Matrix is of dimension 2x2
1       2
3       4

Matrix - 2
The Matrix is of dimension 2x2
4       2
3       1

1.Matrix 1 + Matrix2
2.Matrix 1 x Matrix2
1
The Matrix is of dimension 2x2
5       4
6       5
2
The Matrix is of dimension 2x2
10      4
24      10
Memory Deleted
```

MULTIPLE AND MULTILEVEL INHERITANCE

AIM

Write a C++ program to demonstrate the concept of Multiple and Multilevel inheritance including constructors with parameters.

FLOWCHART



PROGRAM

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

//base class - details of student in CS Department.
class StudentsAtCs{
protected:
    int noOfStudents;
public:
    //constructor for base class.
    StudentsAtCs(int n){
        noOfStudents = n;
    }
    void display(){
        cout<<"No of Students in CS Department"<<setw(14)<<":"<<setw(6)
        <<noOfStudents<<endl;
```

```
    }
};
//base class- details of teachers in CS Department.
class TeachersAtCs{
protected:
    int noOfTeacher;
public:
    //constructor for base class.
    TeachersAtCs(int n){
        noOfTeacher = n;
    }
    void display(){
        cout<<"No of Teachers in CS Department"<<
        setw(14)<<":"<<setw(6)<<noOfTeacher<<endl;
}
};

//multiple inheritance - derived class (whole details of CS Department).
class csDepartment:public StudentsAtCs,public TeachersAtCs{
protected:
    int noOfNonTeachingStaff;
public:
    //constructor for derived class which passes arguments to the
    base class.
    csDepartment(int n1,int n2,int n3):StudentsAtCs(n1),TeachersAtCs(n2){
        noOfNonTeachingStaff=n3;
    }
    void display(){
        cout<<"No of Non Teaching Staff in CS Department"
        <<setw(4)<<":"<<setw(6)<<noOfNonTeachingStaff<<endl;
    }
};
//multilevel inheritance - derived class (whole details in university).
class University:public csDepartment{
    int studentsRegistered;
public:
    University(int n1,int n2,int n3,int n4):csDepartment(n1, n2, n3){
        studentsRegistered=n4;
    }
    void display(){
        StudentsAtCs::display(); //invokes the display function
```

```
        of base class StudentsAtCs
        TeachersAtCs::display(); //invokes the display function
        of base class TeachersAtCs
        csDepartment::display(); //invokes the display function
        of derived class csDepartment
        cout<<"No of Students Registered in the University :
        "<<setw(5)<<studentsRegistered<<endl;
    }
};

int main() {
    cout<<"In this Program there are two base classes , StudentsAtCs
    and TeachersAtCs"<<endl;
    cout<<"which are then inherited by the class csDepartment (multiple inheritance)
    \ncsDepartment is then inherited by class University (multilevel)"<<endl;
    int option;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>option;
    if (option==1) {
        int deptStudents,deptTeachers,deptNonTeachers,UniversityStudents;
        cout<<"No of Students in CS Department"<<endl;
        cin>>deptStudents;
        cout<<"No of Teachers in CS Department"<<endl;
        cin>>deptTeachers;
        cout<<"No of Non Teachers in CS Department"<<endl;
        cin>>deptNonTeachers;

        cout<<"No of Students Registered under the University"<<endl;
        cin>>UniversityStudents;
        University CUSAT(deptStudents,deptTeachers, deptNonTeachers,
        UniversityStudents);
        int choice;
        cout<<"Do you want to display the details\n1.Print\n2.Quit"<<endl;
        cin>>choice;

        if (choice==1) {
            CUSAT.display(); //invokes the display function of the
            derived class University.
        } else {
            return 0;
        }
```

```
    } else {
        cout<<"Successfully Quited !"<<endl;
        return 0;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

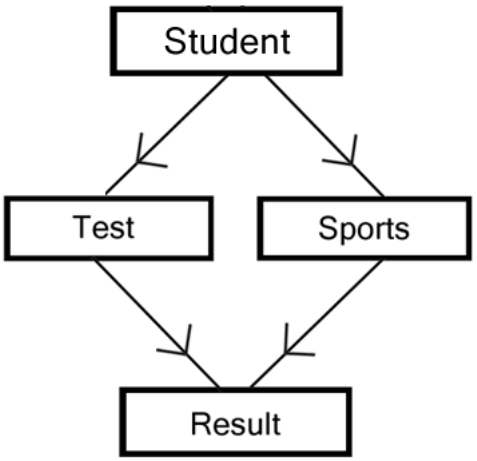
```
In this Program there are two base classes , StudentsAtCs and TeachersAtCs
which are then inherited by the class csDepartment (multiple inheritance)
csDepartment is then inherited by class University (multilevel)
1.Start
2.Quit
1
No of Students in CS Department
2
No of Teachers in CS Department
3
No of Non Teachers in CS Department
4
No of Students Registered under the University
250
Do you want to display the details
1.Print
2.Quit
1
No of Students in CS Department             :     2
No of Teachers in CS Department             :     3
No of Non Teaching Staff in CS Department   :     4
No of Students Registered in the University :   250
```

VIRTUAL BASE CLASS

AIM

Write a C++ program to design a student class representing student roll no. and a test class (derived class of student) representing the scores of the student in various subjects and sports class representing the score in sports. The sports and test class should be inherited by a result class having the functionality to add the scores and display the final result for a student. Demonstrate the concept of Virtual base class on Hybrid inheritance.

FLOWCHART



PROGRAM

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

//base class - Student
class Student{
protected:
    char name[15];
    int rollNo;
public:
    void getRollNo(){
        cout<<"Enter your name"<<endl;
        cin>>name;
```

```
        cout<<"Enter you roll no : ";
        cin>>rollNo;
    }
    void displayRollNo(){
        cout<<"Name"<<setw(16)<<":"<<setw(2)<<name<<endl;
        cout<<"Roll No"<<setw(13)<<":"<<rollNo<<endl;
    }
};

//derived class - Test , inherited Student as virtual base to avoid ambiguity
class Test:public virtual Student{
protected:
    float mark1,mark2;
public:
    void getMarks(){
        cout<<"Enter the mark for subject 1 : ";
        cin>>mark1;
        cout<<"Enter the mark for subject 2 : ";
        cin>>mark2;
    }
    void displayMarks(){
        cout<<"Mark for Subject 1 :"<<mark1<<endl;
        cout<<"Mark for Subject 2 :"<<mark2<<endl;
    }
};

//derived class - Sports , inherited Student as virtual base to avoid ambiguity
class Sports:public virtual Student{
protected:
    int sportsScore;
public:
    void getScore(){
        cout<<"Enter the Score for Sports : ";
        cin>>sportsScore;
    }
    void displayScore(){
        cout<<"Score for Sports"<<setw(4)<<":"<<setw(2)<<sportsScore<<endl;
    }
};

//derived class - Result from Test and Sports.
```

```
class Result:public Test,public Sports{
    float total;
public:
    void totalResult(){
        total = mark1 + mark2 + sportsScore;
    }
    void displayResult(){
        cout<<endl;
        displayRollNo();
        displayMarks();
        displayScore();
        totalResult();
        cout<<"Total"<<setw(16)<<": "<<total<<endl;
    }
};

int main() {
    int option,loopOption;
    cout<<"Welcome\n1.Start\n2.Quit"<<endl;
    cin>>option;
    if (option==1) {
        do {
            Result A;
            A.getRollNo();
            A.getMarks();
            A.getScore();
            A.displayResult();
            cout<<"1.Continue\n2.Quit"<<endl;
            cin>>loopOption;
        } while (loopOption==1);
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

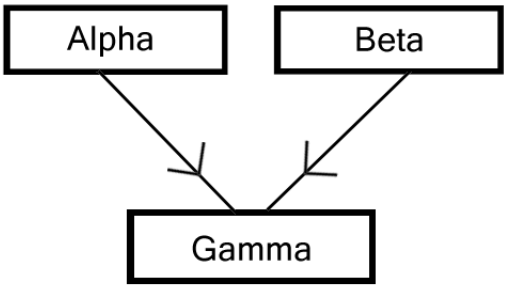
```
Welcome
1.Start
2.Quit
1
Enter your name
Hakkeem
Enter you roll no : 1
Enter the mark for subject 1 : 25
Enter the mark for subject 2 : 75
Enter the Score for Sports : 35

Name               :Hakkeem
Roll No            :1
Mark for Subject 1 :25
Mark for Subject 2 :75
Score for Sports   :35
Total              : 135
```

CONSTRUCTORS DURING INHERITANCE

AIM

Write a C++ program illustrating how the constructors are implemented and the order in which they are called when the classes are inherited. Use three classes named alpha, beta and gamma such that alpha and beta are base classes and gamma is a derived class inheriting alpha & beta.

FLOWCHART



PROGRAM

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

//base class - alpha.
class alpha{
protected:
    int numAlpha;
public:
    //default constructor of alpha.
    alpha(){
        cout<<"Default Constructor of Base Class - Alpha"<<endl;
    }
    //parameterized constructor of alpha.
    alpha(int a){
        numAlpha = a;
        cout<<"Alpha initialized"<<endl;
    }
```

```
    //function to display the value of alpha.
    void displayAlpha(){
        cout<<"Alpha : "<<numAlpha<<endl;
    }
};
//base class - beta.
class beta{
protected:
    int numBeta;
public:
    //default constructor of beta.
    beta(){
        cout<<"Default Constructor of Base Class - Beta"<<endl;
    }
    //parameterized constructor of beta.
    beta(int y){
        numBeta = y;
        cout<<"Beta initialized"<<endl;
    }
    //function to display the value of beta.
    void displayBeta(){
        cout<<"Beta : "<<numBeta<<endl;
    }
};
//derived class gamma from alpha and beta
class gamma:public alpha,public beta{
    int numGamma;
public:
    //default constructor of gamma.
    gamma(){
        cout<<"Default Constructor of Derived Class - Gamma"<<endl;
    }
    //parameterized constructor of gamma which passes arguments to base class.
    gamma(int x,int y,int z):alpha(x),beta(y){
        numGamma = z;
        cout<<"Gamma initialized"<<endl;
    }
    //function to display the value of gamma.
    void displayGamma(){
        cout<<"Gamma : "<<numGamma<<endl;
    }
```

```
};

int main() {
    int choice,Alpha,Beta,Gamma,loopOption;
    cout<<"In this program, two base classes 'Alpha' and 'Beta' is inherited by
    class Gamma"<<endl;
    do {
        cout<<"1.Create Object for Gamma which does not take any arguments
        \n2.Create Object for Gamma with arguments for all the classes
        \n3.Quit"<<endl;
        cin>>choice;
        if(choice == 1){
            //object for gamma which invokes the default constructor.
            gamma obj1;
            cout<<endl;
            cout<<"Even if there are no arguments passed, the default constructor
            of the base classes are
            called\nin the order they are inherited"<<endl;
        }
        else if (choice==2){
            cout<<"Give the input for alpha : ";
            cin>>Alpha;
            cout<<"Give the input for beta : ";
            cin>>Beta;
            cout<<"Give the input for gamma : ";
            cin>>Gamma;
            cout<<endl;
            //object for gamma which invokes the parameterized constructor.
            gamma obj2(Alpha,Beta,Gamma);
            obj2.displayAlpha();
            obj2.displayBeta();
            obj2.displayGamma();
            cout<<"Even though , if the object is created for a derived class as the
            base classes are inherited\ntheir constructor are called in the order in
            which they are inherited";
        }
        cout<<"Do you want to continue?\n1.Continue\n2.Quit"<<endl;
        cin>>loopOption;
    } while (loopOption==1);
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
In this program, two base classes 'Alpha' and 'Beta' is inherited by class Gamma
1.Create Object for Gamma which does not take any arguments
2.Create Object for Gamma with arguments for all the classes
3.Quit
1
Default Constructor of Base Class - Alpha
Default Constructor of Base Class - Beta
Default Constructor of Derived Class - Gamma

Even if there are no arguments passed, the default constructor of the base classes are called
in the order they are inherited
```

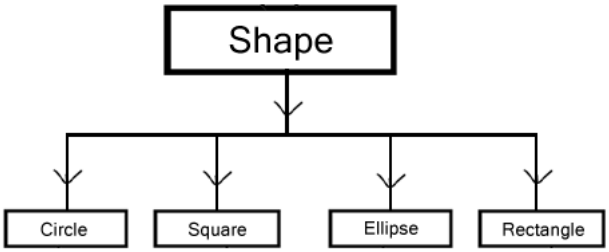
```
2
Give the input for alpha : 20
Give the input for beta : 30
Give the input for gamma : 40

Alpha initialized
Beta initialized
Gamma initialized

Alpha : 20
Beta : 30
Gamma : 40

Even though , if the object is created for a derived class as the base classes are inherited
their constructor are called in the order in which they are inherited
```

RUNTIME POLYMORPHISM

AIM

Write a C++ program to define classes Shapes, Circle, Square, Ellipse and Rectangle with member functions to get the values for finding corresponding areas and print the same. Utilize the concept of Abstract Class and Runtime polymorphism to solve the problem.

FLOWCHART



PROGRAM

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

class Shape{
protected:
    float area;
public:
    //virtual functions , which are redefined in derived classes
    virtual void get_data()=0;
    virtual void get_area()=0;
};

//class Circle inheriting class Shape.
class Circle:public Shape{
    float radius;
public:
    //modifying the function defined in the base class.
    void get_data(){
        cout<<"Enter the Radius of the Circle"<<endl;
        cin>>radius;
    }
    //modifying the function defined in the base class.
```

```
    void get_area(){
        area = (22.0/7.0)*radius*radius;
        cout<<"Area of the Circle is "<<area<<endl;
    }
};

//class Square inheriting class Shape.
class Square:public Shape{
    float side;
public:
    //modifying the function defined in the base class.
    void get_data(){
        cout<<"Enter the Side of the Square"<<endl;
        cin>>side;
    }
//modifying the function defined in the base class.
    void get_area(){
        area = side*side;
        cout<<"Area of the Square is "<<area<<endl;
    }
};

//class Ellipse inheriting class Shape.
class Ellipse:public Shape{
    float minorAxis,majorAxis;
public:
    //modifying the function defined in the base class.
    void get_data(){
        cout<<"Enter the Minor Axis of the Ellipse"<<endl;
        cin>>minorAxis;
        cout<<"Enter the Major Axis of the Ellipse"<<endl;
        cin>>majorAxis;
    }
    //modifying the function defined in the base class.
    void get_area(){
        area = (22.0/7.0)*majorAxis*minorAxis;
        cout<<"The Area of the Ellipse is "<<area<<endl;
    }
};

//class Rectangle inheriting class Shape.
```

```
class Rectangle:public Shape{
    float length,breadth;
public:
    //modifying the function defined in the base class.
    void get_data(){
        cout<<"Enter the Length of the Rectangle"<<endl;
        cin>>length;
        cout<<"Enter the Breadth of the Rectangle"<<endl;
        cin>>breadth;
    }
    //modifying the function defined in the base class.
    void get_area(){
        area=length*breadth;
        cout<<"The Area of the Rectangle is "<<area<<endl;
    }
};
int main() {
    int exitOption,exitLoop;
    cout<<"Welcome\n1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==1) {
        do {
            //pointer of the type base class
            Shape *shapePointer;
            int choiceForArea;
            cout<<"1.Area of Square\n2.Area of Rectangle\n3.Area of Ellipse
            \n4.Area of Circle"<<endl;
            cin>>choiceForArea;
            switch (choiceForArea) {
                case 1:
                    //shapePointer holding the address of the the object of class
                    Square.
                    shapePointer = new Square;
                    //invokes the function of the class Square.
                    shapePointer->get_data();
                    shapePointer->get_area();
                    break;
                case 2:
                    //shapePointer holding the address of the the object of class
                    Rectangle.
                    shapePointer = new Rectangle;
```

```
                    //invokes the function of the class Rectangle.
                    shapePointer->get_data();
                    shapePointer->get_area();
                    break;
                case 3:
                    //shapePointer holding the address of the the object of class
                    Ellipse.
                    shapePointer = new Ellipse;
                    shapePointer->get_data();
                    shapePointer->get_area();
                    break;
                case 4:
                    //shapePointer holding the address of the the object of class
                    Circle.
                    shapePointer = new Circle;
                    //invokes the function of the class Circle.
                    shapePointer->get_data();
                    shapePointer->get_area();
                    break;
                default:
                    break;
            }
            cout<<"Do you want to continue?\n1.Continue\n2.Quit"<<endl;
            cin>>exitLoop;
        } while (exitLoop==1);
    }
    if(exitLoop!=1 or exitOption!=1){
        cout<<"Successfully Exited!"<<endl;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Welcome
1.Start
2.Quit
1
1.Area of Square
2.Area of Rectangle
3.Area of Ellipse
4.Area of Circle
1
Enter the Side of the Square
4
Area of the Square is 16
```

```
2
Enter the Length of the Rectangle
3
Enter the Breadth of the Rectangle
6
The Area of the Rectangle is 18
```

```
3
Enter the Minor Axis of the Ellipse
4
Enter the Major Axis of the Ellipse
5
The Area of the Ellipse is 62.8571
```

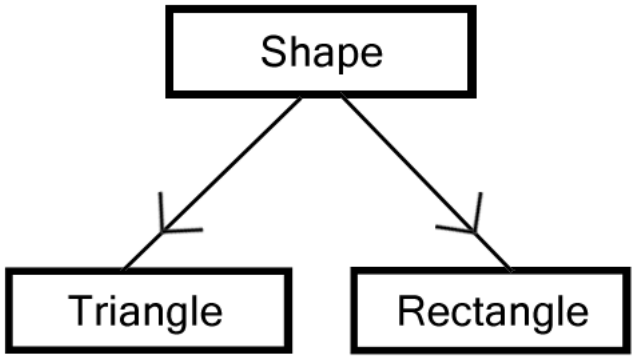
```
4
Enter the Radius of the Circle
6
Area of the Circle is 113.143
```

PURE VIRTUAL FUNCTIONS

AIM

Write a C++ program to demonstrate the use of pure virtual functions and abstract base classes.

FLOWCHART



PROGRAM

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

class Shape{
protected:
    double dimension1,dimension2;
public:
    void get_data(double a,double b){
        dimension1=a;
        dimension2=b;
    }
    virtual void display_area()=0;
};
class Triangle:public Shape{
    double areaOfTriangle;
public:
```

```
    void display_area(){
        areaOfTriangle=0.5*dimension1*dimension2;
        cout<<"Area of the Triangle : "<<areaOfTriangle<<endl;
    }
};
class Rectangle:public Shape{
    double areaOfRectangle;
public:
    void display_area(){
        areaOfRectangle=dimension1*dimension2;
        cout<<"Area of the Rectangle : "<<areaOfRectangle<<endl;
    }
};

int main() {
    int exitOption,loopOption;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==1) {
        do {
            int choice;
            Shape *shapePointer;
            cout<<"1.Triangle\n2.Rectangle\n3.Quit"<<endl;
            cin>>choice;
            switch (choice) {
                case 1:
                    double base,height;
                    cout<<"Enter the base and height"<<endl;
                    cin>>base>>height;
                    shapePointer = new Triangle;
                    shapePointer->get_data(base, height);
                    shapePointer->display_area();
                    break;
                case 2:
                    double length,breadth;
                    cout<<"Enter the length and breadth"<<endl;
                    cin>>length>>breadth;
                    shapePointer = new Rectangle;
                    shapePointer->get_data(length,breadth);
                    shapePointer->display_area();
                    break;
```

```
                default:
                    return 0;
            }
            cout<<"Do you want to continue?\n1.Continue\n2.Quit"<<endl;
            cin>>loopOption;
        } while (loopOption==1);
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
1.Triangle
2.Rectangle
3.Quit
1
Enter the base and height
2 3
Area of the Triangle : 3
```

```
2
Enter the length and breadth
2 5
Area of the Rectangle : 10
```

TEMPLATE CLASS

AIM

Write a C++ Program to demonstrate the use of class templates.

PROGRAM

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

template <class T>
class Calculator{
    T num1,num2;
public:
    Calculator(T a,T b){
        num1=a;
        num2=b;
    }
    T sum(){
        T sum;
        sum = num1+num2;
        return sum;
    }
    T subtract(){
        T difference;
        difference = num1-num2;
        return difference;
    }
    T multiply(){
        T product;
        product = num1*num2;
        return product;
    }
    T divide(){
        T ratio;
        ratio = num1/num2;
        return ratio;
    }
};
int main() {
    int choice;
```

```
cout<<"1.Integer Operations\n2.Decimal Operations\n3.Quit"<<endl;
cin>>choice;
if (choice==1) {
    int num1,num2;
    cout<<"Enter the numbers"<<endl;
    cin>>num1>>num2;
    Calculator <int> obj1(num1,num2);
    int loopOption;
    do {
        cout<<"1.Addition\n2.Subtraction\n3.Multiplication\n4.Division"<<endl;
        int key;
        cin>>key;
        switch (key) {
            case 1:
                cout<<num1<<" + "<<num2<<" = "<<obj1.sum()<<endl;
                break;
            case 2:
                cout<<num1<<" - "<<num2<<" = "<<obj1.subtract()<<endl;
                break;
            case 3:
                cout<<num1<<" x "<<num2<<" = "<<obj1.multiply()<<endl;
                break;
            case 4:
                cout<<num1<<" / "<<num2<<" = "<<obj1.divide()<<endl;
                break;
            default:
                break;
        }
        cout<<"Do you want to continue\n1.Continue\n2.Exit"<<endl;
        cin>>loopOption;
    } while (loopOption==1);
}
else if (choice==2) {
    float num1,num2;
    cout<<"Enter the numbers"<<endl;
    cin>>num1>>num2;
    Calculator <float> obj2(num1,num2);
    int loopOption;
    do {
        cout<<"1.Addition\n2.Subtraction\n3.Multiplication\n4.Division"<<endl;
        int key;
```

```
            cin>>key;
            switch (key) {
                case 1:
                    cout<<num1<<" + "<<num2<<" = "<<obj2.sum()<<endl;
                    break;
                case 2:
                    cout<<num1<<" - "<<num2<<" = "<<obj2.subtract()<<endl;
                    break;
                case 3:
                    cout<<num1<<" x "<<num2<<" = "<<obj2.multiply()<<endl;
                    break;
                case 4:
                    cout<<num1<<" / "<<num2<<" = "<<obj2.divide()<<endl;
                    break;
                default:
                    break;
            }
            cout<<"Do you want to continue\n1.Continue\n2.Exit"<<endl;
            cin>>loopOption;
        } while (loopOption==1);
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Integer Operations
2.Decimal Operations
3.Quit
1
Enter the numbers
2 3
1.Addition
2.Subtraction
3.Multiplication
4.Division
1
2 + 3 = 5
2
2 - 3 = -1
3
2 x 3 = 6
4
2 / 3 = 0
```

```
1.Integer Operations
2.Decimal Operations
3.Quit
2
Enter the numbers
2.5 5.2
1.Addition
2.Subtraction
3.Multiplication
4.Division
1
2.5 + 5.2 = 7.7
2
2.5 - 5.2 = -2.7
3
2.5 x 5.2 = 13
4
2.5 / 5.2 = 0.480769
```

EXCEPTION HANDLING

AIM

Write a C++ Program to demonstrate the use of exception handling.

PROGRAM

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

double computeArea(float a,float b,float c){
    //if the triangle inequality theorem is not satisfied it throws an error.
    if((a+b)<=c or (b+c)<=a or (a+c)<=b){
        //throws the error message and function will be terminated.
        throw "These dimensions does not satisfy a triangle";
    }
    //else the area will be computed.
    else {
        double s,area;
        s=(a+b+c)/2;
        area=s*(s-a)*(s-b)*(s-c);
        if (area<0){
            area = area*-1;
        }
        area = sqrt(area);
        return area;
    }
}

int main() {
    int exitOption,loopOption;
    cout<<"1.Start\n2.Quit"<<endl;
    cin>>exitOption;
    if (exitOption==1) {
        float side1,side2,side3;
        do {
            cout<<"Enter the sides of a triangle"<<endl;
            cin>>side1>>side2>>side3;
            try {
                double area;
```

```
                area= computeArea(side1,side2,side3);
                cout<<"Area is "<<area<<endl;
                cout<<"Do you want to continue?\n1.Continue\n2.Quit"<<endl;
                cin>>loopOption;
            } catch (const char *message) {
                cout<<message<<endl;
                cout<<"Do you want to continue?\n1.Continue\n2.Quit"<<endl;
                cin>>loopOption;
            }
        } while (loopOption==1);
    } else {
        cout<<"Successfully Quited"<<endl;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
1.Start
2.Quit
1
Enter the sides of a triangle
1 2 3
These dimensions does not satisfy a triangle
Enter the sides of a triangle
3 4 5
Area is 6
```