**1. Design a class Matrix of dimension 3x3. Overload + operator to find sum of two matrices.**

#include <iostream>

using namespace std;

class Matrix {

private:

int mat[3][3];

public:

void input() {

cout << "Enter 3x3 matrix elements:\n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> mat[i][j];

}

void display() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++)

cout << mat[i][j] << " ";

cout << endl;

}

}

Matrix operator+(const Matrix& m) {

Matrix result;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

result.mat[i][j] = mat[i][j] + m.mat[i][j];

return result;

}

};

int main() {

Matrix m1, m2, m3;

cout << "For Matrix 1:\n";

m1.input();

cout << "For Matrix 2:\n";

m2.input();

cout << "Matrix 1:\n";

m1.display();

cout << "Matrix 2:\n";

m2.display();

m3 = m1 + m2;

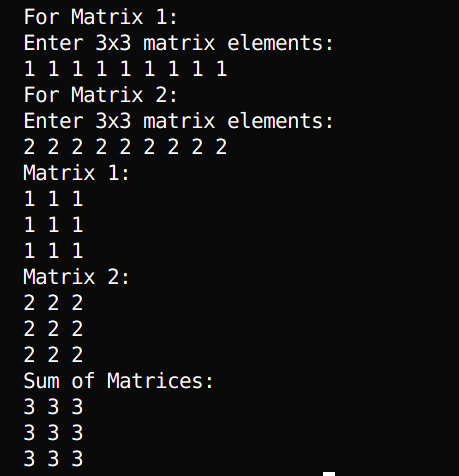
cout << "Sum of Matrices:\n";

m3.display();

return 0;

}

**OUTPUT**



**2. Define a class string and use + and > operators to concatenate and compare two strings respectively.**

#include <iostream>

#include <cstring>

using namespace std;

class String {

private:

char str[100];

public:

String() {

str[0] = '\0';

}

String(const char\* s) {

strcpy(str, s);

}

void input() {

cout << "Enter string: ";

cin >> str;

}

void display() {

cout << str << endl;

}

String operator+(const String& s) {

String result;

strcpy(result.str, str);

strcat(result.str, s.str);

return result;

}

bool operator>(const String& s) {

return strcmp(str, s.str) > 0;

}

};

int main() {

String s1, s2, s3;

cout << "For String 1:\n";

s1.input();

cout << "For String 2:\n";

s2.input();

cout << "String 1: ";

s1.display();

cout << "String 2: ";

s2.display();

s3 = s1 + s2;

cout << "Concatenated String: ";

s3.display();

if (s1 > s2)

cout << "String 1 is greater than String 2" << endl;

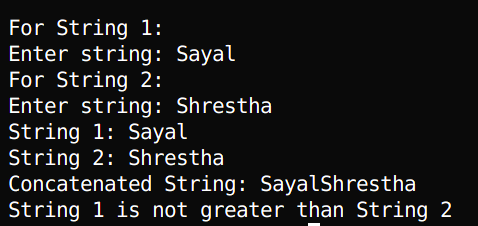
else

cout << "String 1 is not greater than String 2" << endl;

return 0;

}

**OUTPUT**



**3. Write a program to implement vector addition and subtraction using operator overloading.**

#include <iostream>

using namespace std;

class Vector {

private:

double x, y, z;

public:

Vector(double xCoord = 0, double yCoord = 0, double zCoord = 0) : x(xCoord), y(yCoord),

z(zCoord) {}

void input() {

cout << "Enter x, y, z components: ";

cin >> x >> y >> z;

}

void display() {

cout << "(" << x << ", " << y << ", " << z << ")" << endl;

}

Vector operator+(const Vector& v) {

return Vector(x + v.x, y + v.y, z + v.z);

}

Vector operator-(const Vector& v) {

return Vector(x - v.x, y - v.y, z - v.z);

}

};

int main() {

Vector v1, v2, v3, v4;

cout << "For Vector 1:\n";

v1.input();

cout << "For Vector 2:\n";

v2.input();

cout << "Vector 1: ";

v1.display();

cout << "Vector 2: ";

v2.display();

v3 = v1 + v2;

v4 = v1 - v2;

cout << "Addition Result: ";

v3.display();

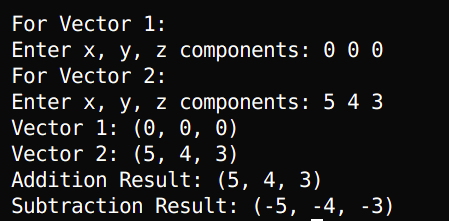
cout << "Subtraction Result: ";

v4.display();

return 0;

}

**OUTPUT**



**4. Design a class Matrix, overload ++ and -- operator to increment and decrement each element of**

**the matrix by 1**

#include <iostream>

using namespace std;

class Matrix {

private:

int mat[3][3];

public:

Matrix() {

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j] = 0;

}

void input() {

cout << "Enter 3x3 matrix elements:\n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> mat[i][j];

}

void display() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++)

cout << mat[i][j] << " ";

cout << endl;

}

}

Matrix& operator++() { // Prefix increment

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]++;

return \*this;

}

Matrix& operator--() { // Prefix decrement

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]--;

return \*this;

}

Matrix operator++(int) { // Postfix increment

Matrix temp = \*this;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]++;

return temp;

}

Matrix operator--(int) { // Postfix decrement

Matrix temp = \*this;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

mat[i][j]--;

return temp;

}

};

int main() {

Matrix m;

cout << "Enter matrix elements:\n";

m.input();

cout << "Original Matrix:\n";

m.display();

cout << "After Prefix ++:\n";

++m;

m.display();

cout << "After Postfix ++:\n";

m++;

m.display();

cout << "After Prefix --:\n";

--m;

m.display();

cout << "After Postfix --:\n";

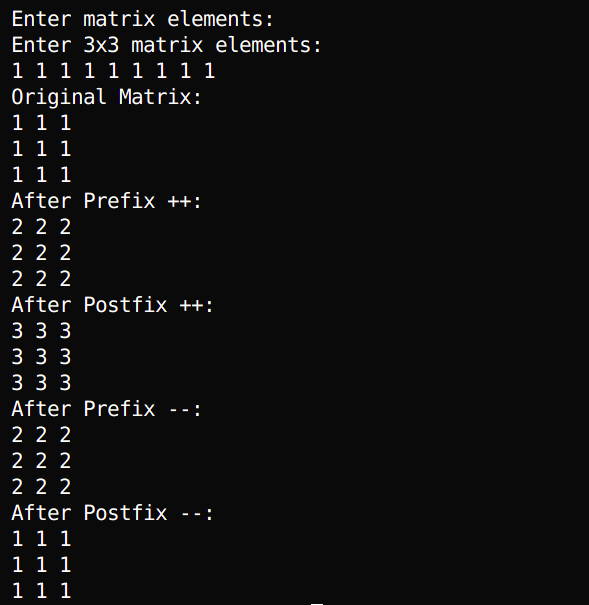
m--;

m.display();

return 0;

}

**OUTPUT**



**5. Write a program to access elements of a vector class with index operator.**

#include <iostream>

using namespace std;

class Vector {

private:

int arr[5]; // Fixed size vector of 5 elements

int size;

public:

Vector() : size(5) {

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

arr[i] = 0;

}

void input() {

cout << "Enter " << size << " elements: ";

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

cin >> arr[i];

}

void display() {

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

cout << arr[i] << " ";

cout << endl;

}

int& operator[](int index) {

if (index >= 0 && index < size)

return arr[index];

cout << "Index out of bounds!" << endl;

return arr[0]; // Return first element as default for invalid index

}

};

int main() {

Vector v;

cout << "Input vector elements:\n";

v.input();

cout << "Original Vector: ";

v.display();

cout << "Accessing element at index 2: " << v[2] << endl;

v[2] = 100; // Modify element using index operator

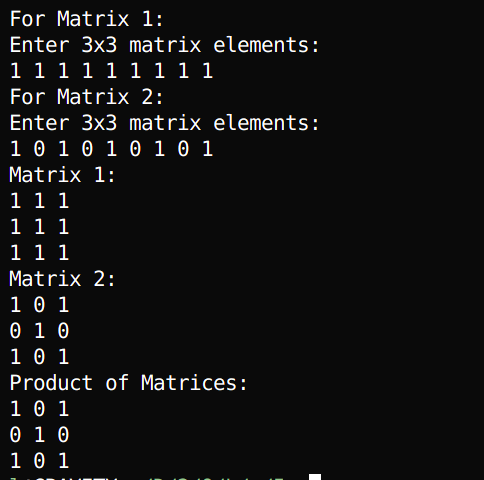
cout << "Vector after modifying index 2: ";

v.display();

return 0;

}

**OUTPUT**



**6. Write a program to muliply two matrices by overloading the \* operator.**

#include <iostream>

using namespace std;

class Matrix {

private:

int mat[3][3];

public:

void input() {

cout << "Enter 3x3 matrix elements:\n";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

cin >> mat[i][j];

}

void display() {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++)

cout << mat[i][j] << " ";

cout << endl;

}

}

Matrix operator\*(const Matrix& m) {

Matrix result;

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

result.mat[i][j] = mat[i][j] \* m.mat[i][j];

return result;

}

};

int main() {

Matrix m1, m2, m3;

cout << "For Matrix 1:\n";

m1.input();

cout << "For Matrix 2:\n";

m2.input();

cout << "Matrix 1:\n";

m1.display();

cout << "Matrix 2:\n";

m2.display();

m3 = m1 \* m2;

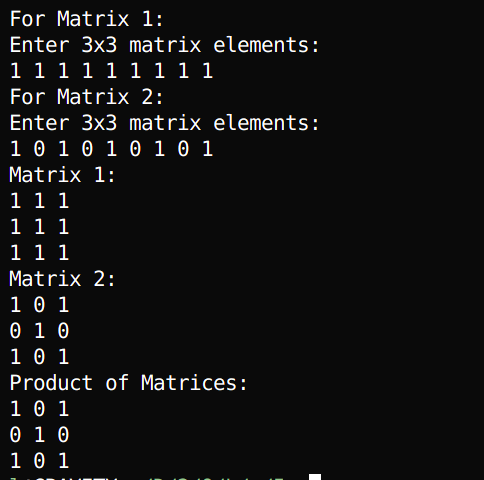
cout << "Product of Matrices:\n";

m3.display();

return 0;

}

**OUTPUT**



**7. Create a class named City that will have two member variables CityName and DistFromKtm (float). Add member functions to set and retrieve the CityName and DistFromKtm separately. Add operator overloading to find the distance between the cities (just find the difference of DistFromKtm) and sum of distance of those cities from Kathmandu. In the main function, initialize three city objects. Set the first and second city to be Pokhara and Dhangadi. Display the sum of DistFromKtm of Pokhara and Dhangadi and distance between Pokhara and Dhangadi.**

#include<iostream>

#include<string>

using namespace std;

class City {

private:

string CityName;

float DistFromKtm;

public:

void setCityName(string name) {

CityName = name;

}

void setDistFromKtm(float dist) {

DistFromKtm = dist;

}

string getCityName() {

return CityName;

}

float getDistFromKtm() {

return DistFromKtm;

}

// Operator overloading

float operator+(City c) {

return this->DistFromKtm + c.DistFromKtm;

}

float operator-(City c) {

return this->DistFromKtm - c.DistFromKtm;

}

};

int main() {

City c1, c2, c3;

float dist;

c1.setCityName("Pokhara");

cout << "Enter distance of Pokhara from Kathmandu: ";

cin >> dist;

c1.setDistFromKtm(dist);

c2.setCityName("Dhangadi");

cout << "Enter distance of Dhangadi from Kathmandu: ";

cin >> dist;

c2.setDistFromKtm(dist);

cout << "\nSum of distances from Kathmandu: " << c1 + c2 << " km" << endl;

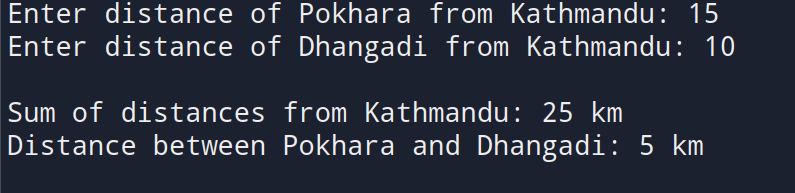
cout << "Distance between " << c1.getCityName() << " and " << c2.getCityName()

<< ": " << abs(c1 - c2) << " km" << endl;

return 0;

}

**OUTPUT**



**8. Write a program to overload the relational operators to compare the length (in meter and centimeter) of two objects.**

#include <iostream>

#include <cstring>

using namespace std;

class length {

private:

int m=0;

int cm;

public:

void input() {

cout << "Enter centimeter: ";

cin >> cm;

}

void display() {

cout << m << "m " << cm << "cm" << endl;

}

void convert() {

m += cm / 100;

cm = cm % 100;

}

bool operator>(const length& l) {

return cm > l.cm;

}

};

int main() {

length l1, l2, l3;

cout << "For length 1:\n";

l1.input();

cout << "For length 2:\n";

l2.input();

if (l1 > l2)

{cout << "length 1 is greater than length 2" << endl;

cout << "length 1: ";

l1.convert();

l1.display();

cout << "length 2: ";

l2.convert();

l2.display();}

else

{cout << "length 1 is not greater than length 2" << endl;

cout << "length 1: ";

l1.convert();

l1.display();

cout << "length 2: ";

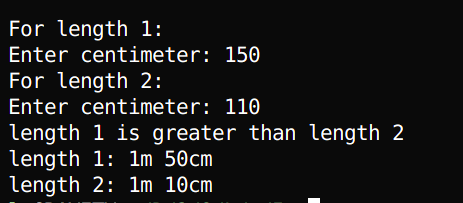
l2.convert();

l2.display();}

return 0;

}

**OUTPUT**



**9. Create a class called time that has separate int member data for hours, minutes, and seconds. One constructor should initialize this data to zero (0), and another should initialize it to fixed values. A member function should display it in 10:45:30 format. The final member function should add two objects of type time passed as arguments using operator overloading.**

#include <iostream>

#include <iomanip>

using namespace std;

class Time {

private:

int days;

int hours;

int minutes;

int seconds;

public:

Time() {

days = 0;

hours = 0;

minutes = 0;

seconds = 0;

}

void getTime() {

cout << "Enter hours: ";

cin >> hours;

cout << "Enter minutes: ";

cin >> minutes;

cout << "Enter seconds: ";

cin >> seconds;

normalize();

}

void normalize() {

minutes += seconds / 60;

seconds = seconds % 60;

hours += minutes / 60;

minutes = minutes % 60;

days += hours / 24;

hours = hours % 24;

}

friend ostream& operator<<(ostream& os, const Time& t) {

if (t.days > 0) {

os << t.days << " day";

if (t.days > 1) os << "s";

os << ", ";

}

os << t.hours << ":"

<< setw(2) << setfill('0') << t.minutes << ":"

<< setw(2) << setfill('0') << t.seconds;

return os;

}

Time operator+(Time& t2) {

Time result;

result.seconds = seconds + t2.seconds;

result.minutes = minutes + t2.minutes;

result.hours = hours + t2.hours;

result.days = days + t2.days;

result.normalize();

return result;

}

bool operator>=(Time& t2) {

int totalSeconds1 = days \* 24 \* 60 \* 60 + hours \* 60 \* 60 + minutes \* 60 + seconds;

int totalSeconds2 = t2.days \* 24 \* 60 \* 60 + t2.hours \* 60 \* 60 + t2.minutes \* 60 + t2.seconds;

return totalSeconds1 >= totalSeconds2;

}

bool operator<=(Time& t2) {

int totalSeconds1 = days \* 24 \* 60 \* 60 + hours \* 60 \* 60 + minutes \* 60 + seconds;

int totalSeconds2 = t2.days \* 24 \* 60 \* 60 + t2.hours \* 60 \* 60 + t2.minutes \* 60 + t2.seconds;

return totalSeconds1 <= totalSeconds2;

}

};

int main() {

Time t1, t2, sum;

cout << "Enter first time:" << endl;

t1.getTime();

cout << "\nEnter second time:" << endl;

t2.getTime();

sum = t1 + t2;

cout << "\nTime 1: ";

cout << t1;

cout << "\nTime 2: ";

cout << t2;

cout << "\nSum: ";

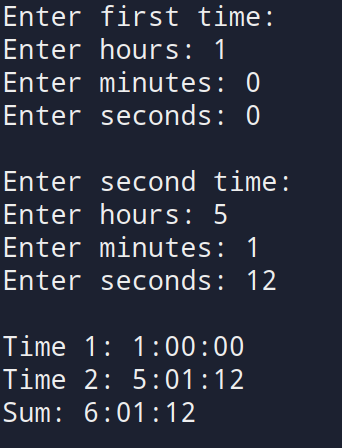
cout << sum;

cout<<endl;

return 0;

}

**OUTPUT**



# DISCUSSION

This lab focused on implementing operator overloading in C++ using both member and friend functions. We performed arithmetic and relational operations on user-defined types like Matrix, Vector, String, City, and Time. Each example showed how familiar operators can be adapted to work intuitively with objects. For instance, we overloaded + to add matrices and vectors, > to compare strings and lengths, and [] to access vector elements. Both unary (++, --) and binary operators were used effectively, helping us understand how operator behavior can be customized for user-defined classes.

# CONCLUSION

This lab helped us understand the concept and significance of operator overloading in C++. We practiced overloading both unary and binary operators using member and friend functions, aligning with the objective of enabling intuitive operations on user-defined types. By working with classes like Matrix, Vector, String, City, and Time, we saw how overloading improves code readability, supports abstraction, and promotes reusability. Overall, the lab reinforced the importance of operator overloading as a form of compile-time polymorphism in object-oriented programming.