**Logo, company name

Description automatically generated**

**Department of (Computer Science)**

**Pak-Austria** Fachhochschule**: Institute of Applied Sciences and Technology, Haripur, Pakistan**

**COMP-112L Object Oriented Programming Lab**

**Lab Journal**

**Class: BS Computer Science**

**Name: Ahmed Raza**

**Registration No.: B20F0436CS031**

**Semester: 3rd**

**Submission Date: 2nd December, 2021**

**Submitted to: Engr. Rafi-Ullah**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructor Signature**

**Lab No. 07**

**Operating Overloading**

**Objectives:**

In this lab we will be discussing about Operator Overloading in detail. This is one of the most important concepts in Object Orientation. In C++ the overloading principle applies not only to functions, but to operators too. That is, of operators can be extended to work not just with built-in types but also classes.

**Tools/Software Required:**

* All the tasks are implemented on DEV C++.

**Introduction:**

**Operator Overloading**

A programmer can provide his or her own operator to a class by overloading the built-in operator to perform some specific computation when the operator is used on objects of that class. On the other hand, operator overloading, like any advanced C++ feature, makes the language more complicated. In addition, operators tend to have very specific meaning, and most programmers don't expect operators to do a lot of work, so overloading operators can be abused to make code unreadable.

**Assignment operator**

The assignment operator can be overloaded. We did not have to call any accessor functions in order to get the real and imaginary parts from the parameter other since the overloaded operator is a member of the class and has full access to all private data. Alternatively, we could have defined the addition operator globally and called a member to do the actual work.

**Lab Tasks:**

**Task # 01:**

Write a C++ program for the matric multiplication using operator overloading. Overload the operator (\*) and solve matrix multiplication. Here are some key consideration that you keep in mind while developing solution.

In this task you will take a 3x3 matrix.

**Description**:

* Class Matrix has a constructor that will initialize its element to 0.
* Also has a set () function that can be used to set values in the Matrix.
* A show () function shows the matrix.
* Operator overloading definition for "\*" operator.

**Code:**

**#include<iostream>**

**using namespace std;**

**class Matrix**

**{**

**private :**

**int a11;int a12;int a13;**

**int a21;int a22;int a23;**

**int a31;int a32;int a33;**

**int b11;int b12;int b13;**

**int b21;int b22;int b23;**

**int b31;int b32;int b33;**

**int z11;int z12;int z13;**

**int z21;int z22;int z23;**

**int z31;int z32;int z33;**

**public:**

**int MUL(int a[][3], int b[][3], int mul[][3])**

**{**

**//Initializing the all indexes of Mul matrix**

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

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

**{**

**mul[i][j]=0;**

**}**

**// Multiplying the both matrix to get result in Mul Matrix.**

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

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

**for(int k = 0; k<3; ++k)**

**{**

**mul[i][j] += a[i][k] \* b[k][j];**

**}**

**return mul[3][3];**

**}**

**// SET FOR A**

**void set\_a11(int c11){a11=c11;}**

**void set\_a12(int c12){a12=c12;}**

**void set\_a13(int c13){a13=c13;}**

**void set\_a21(int c21){a21=c21;}**

**void set\_a22(int c22){a22=c22;}**

**void set\_a23(int c23){a23=c23;}**

**void set\_a31(int c31){a31=c31;}**

**void set\_a32(int c32){a32=c32;}**

**void set\_a33(int c33){a33=c33;}**

**// SET FOR B**

**void set\_b11(int d11){b11=d11;}**

**void set\_b12(int d12){b12=d12;}**

**void set\_b13(int d13){b13=d13;}**

**void set\_b21(int d21){b21=d21;}**

**void set\_b22(int d22){b22=d22;}**

**void set\_b23(int d23){b23=d23;}**

**void set\_b31(int d31){b31=d31;}**

**void set\_b32(int d32){b32=d32;}**

**void set\_b33(int d33){b33=d33;}**

**// SET FOR Z**

**void set\_z11(int y11){z11=y11;}**

**void set\_z12(int y12){z12=y12;}**

**void set\_z13(int y13){z13=y13;}**

**void set\_z21(int y21){z21=y21;}**

**void set\_z22(int y22){z22=y22;}**

**void set\_z23(int y23){z23=y23;}**

**void set\_z31(int y31){z31=y31;}**

**void set\_z32(int y32){z32=y32;}**

**void set\_z33(int y33){z33=y33;}**

**//GET FOR A**

**int get\_a11(){return a11;}**

**int get\_a12(){return a12;}**

**int get\_a13(){return a13;}**

**int get\_a21(){return a21;}**

**int get\_a22(){return a22;}**

**int get\_a23(){return a23;}**

**int get\_a31(){return a31;}**

**int get\_a32(){return a32;}**

**int get\_a33(){return a33;}**

**//GET FOR B**

**int get\_b11(){return b11;}**

**int get\_b12(){return b12;}**

**int get\_b13(){return b13;}**

**int get\_b21(){return b21;}**

**int get\_b22(){return b22;}**

**int get\_b23(){return b23;}**

**int get\_b31(){return b31;}**

**int get\_b32(){return b32;}**

**int get\_b33(){return b33;}**

**// GET FOR Z**

**int get\_z11(){return z11;}**

**int get\_z12(){return z12;}**

**int get\_z13(){return z13;}**

**int get\_z21(){return z21;}**

**int get\_z22(){return z22;}**

**int get\_z23(){return z23;}**

**int get\_z31(){return z31;}**

**int get\_z32(){return z32;}**

**int get\_z33(){return z33;}**

**};**

**int main()**

**{**

**Matrix C1;**

**int a[3][3];**

**int b[3][3];**

**int mul[3][3];**

**int r11,r12,r13,r21,r22,r23,r31,r32,r33;**

**int t11,t12,t13,t21,t22,t23,t31,t32,t33;**

**int result11,result12,result13,result21,result22,result23,result31,result32,result33;**

**cout<<"FOR THE FIRST MATRIX "<<endl;**

**for(int i=1; i<=3; i++)**

**{**

**for(int j=1; j<=3; j++)**

**{**

**cout<<"Enter value in a"<<i<<j<<" : ";**

**cin>>a[i-1][j-1];**

**cout<<endl;**

**}**

**}**

**cout<<"FOR THE SECOND MATRIX "<<endl;**

**for(int i=1; i<=3; i++)**

**{**

**for(int j=1; j<=3; j++)**

**{**

**cout<<"Enter value in a"<<i<<j<<" : ";**

**cin>>b[i-1][j-1];**

**cout<<endl;**

**}**

**}**

**//Set for A**

**C1.set\_a11(a[0][0]);C1.set\_a12(a[0][1]);C1.set\_a13(a[0][2]);**

**C1.set\_a21(a[1][0]);C1.set\_a22(a[1][1]);C1.set\_a23(a[1][2]);**

**C1.set\_a31(a[2][0]);C1.set\_a32(a[2][1]);C1.set\_a33(a[2][2]);**

**//Set for B**

**C1.set\_b11(b[0][0]);C1.set\_b12(b[0][1]);C1.set\_b13(b[0][2]);**

**C1.set\_b21(b[1][0]);C1.set\_b22(b[1][1]);C1.set\_b23(b[1][2]);**

**C1.set\_b31(b[2][0]);C1.set\_b32(b[2][1]);C1.set\_b33(b[2][2]);**

**C1.MUL(a,b,mul);**

**C1.set\_z11(mul[0][0]);C1.set\_z12(mul[0][1]);C1.set\_z13(mul[0][2]);**

**C1.set\_z21(mul[1][0]);C1.set\_z22(mul[1][1]);C1.set\_z23(mul[1][2]);**

**C1.set\_z31(mul[2][0]);C1.set\_z32(mul[2][1]);C1.set\_z33(mul[2][2]);**

**// Getting values from array A**

**r11=C1.get\_a11();r12=C1.get\_a12();r13=C1.get\_a13();**

**r21=C1.get\_a21();r22=C1.get\_a22();r23=C1.get\_a23();**

**r31=C1.get\_a31();r32=C1.get\_a32();r33=C1.get\_a33();**

**// Getting values from array B**

**t11=C1.get\_b11();t12=C1.get\_b12();t13=C1.get\_b13();**

**t21=C1.get\_b21();t22=C1.get\_b22();t23=C1.get\_b23();**

**t31=C1.get\_b31();t32=C1.get\_b32();t33=C1.get\_b33();**

**// Getting values from array C**

**result11=C1.get\_z11();result12=C1.get\_z12();result13=C1.get\_z13();**

**result21=C1.get\_z21();result22=C1.get\_z22();result23=C1.get\_z23();**

**result31=C1.get\_z31();result32=C1.get\_z32();result33=C1.get\_z33();**

**cout<<"\tFIRST MATRIX IS "<<endl;**

**cout<<"\t"<<r11<<"\t"<<r12<<"\t"<<r13<<endl;**

**cout<<"\t"<<r21<<"\t"<<r22<<"\t"<<r23<<endl;**

**cout<<"\t"<<r31<<"\t"<<r32<<"\t"<<r33<<endl;**

**cout<<"\tSECOND MATRIX IS "<<endl;**

**cout<<"\t"<<t11<<"\t"<<t12<<"\t"<<t13<<endl;**

**cout<<"\t"<<t21<<"\t"<<t22<<"\t"<<t23<<endl;**

**cout<<"\t"<<t31<<"\t"<<t32<<"\t"<<t33<<endl;**

**cout<<"\tMULTIPLE MATRIX IS "<<endl;**

**cout<<"\t"<<result11<<"\t"<<result12<<"\t"<<result13<<endl;**

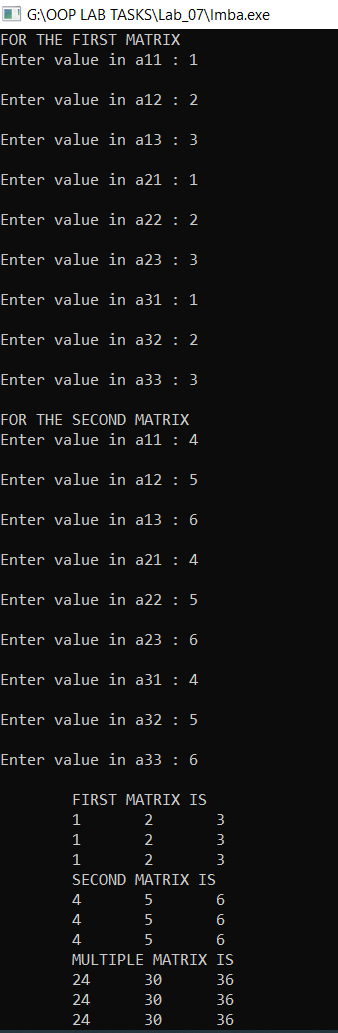
**cout<<"\t"<<result21<<"\t"<<result22<<"\t"<<result23<<endl;**

**cout<<"\t"<<result31<<"\t"<<result32<<"\t"<<result33;**

**return 0;**

**}**

**Output:**

****

**Task #02:**

Write a C++ program to overload the plus operator by using the concept of operator overloading. There must be a class having default constructor. Create three objects of the class and store the addition of two objects in third object, also include print function to print the variables.

**Code:**

**#include<iostream>**

**using namespace std;**

**class Oper\_Overload {**

**int num;**

**public:**

**void set\_num(int s) {**

**num = s;**

**}**

**Oper\_Overload operator+(Oper\_Overload ob) {**

**Oper\_Overload t;**

**t.num = num + ob.num;**

**return (t);**

**}**

**void display() {**

**cout << num << endl;**

**}**

**};**

**int main() {**

**Oper\_Overload num1, num2, result;**

**int a, b;**

**cout << "Enter the value of Complex Numbers a: ";**

**cin >> a;**

**cout << "Enter the value of Complex Numbers b: ";**

**cin>>b;**

**num1.set\_num(a);**

**num2.set\_num(b);**

**result = num1 + num2 ;**

**cout << "Input Values:\n";**

**num1.display();**

**num2.display();**

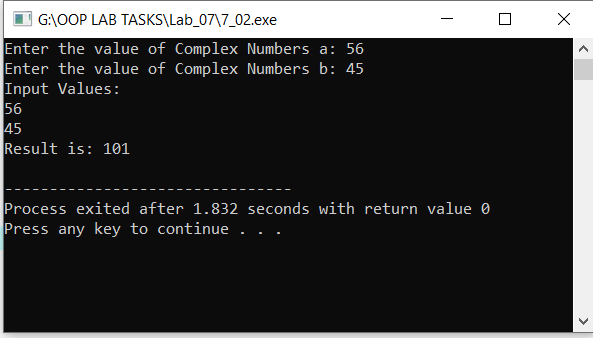
**cout << "Result is: ";**

**result.display();**

**return 0;**

**}**

**Output:**

****

**Results & Observations:**

In this Lab I’ve learned about the concept of operator overloading & also I understand the concept of friend function that how can we use friend function in the class. As well as I learned about static keywords and static member variables. In the first task, I’ve used a Class with name Matrixin which I’ve used Mul function that’s taking two arrays from the main function and multiply both and return it after saving it in the third array. In the second task, I’ve used the operator overloading concept then in the main function I’ve created three objects of the Oper-Overload class through which I’ve accessed data from the class.