OSU Mechanical Engineering

Smart Products Laboratory

Software II | Two

ME Course Number
Spring 2019

Table of Contents

List of Figures

1 Overview

This lab will work on the use of classes in C++ to implement object-oriented programming. We will create a Matrix class.

2 Background

The form below provides a general template for creating classes.

2.1 C++ for Object Oriented Programming

A class has a couple different types of internal elements. The main ones are,

- Constructors/Destructors
- Operator overloading
- Members (non-functional properties)
- Methods (functional)

These are briefly demonstrated in the following coding example, feel free to use this as a helpful reference when creating your classes.

```
//_Title: ClassName.h

//Always put this preprocessor declaration in your header files to avoid
// multiple loading of the same file.
#ifndef CLASSNAME_H
#define CLASSNAME_H
#include <iostream> // use < >  when it is a standard lib
#include "NonStandardLib.h" // use "filename.h" when you have self-defined files.

// --------------Namespace-----------------------------
// creating your own namespace
//------------------------------------------------------
namespace myNamespace {
class ClassName: public ParentClass
{
        // --------------Inheritance---------------------------------
        // ClassName now inherits all capabilities of ParentClass
        //------------------------------------------------------------
private:
        // --------------Private Elements-----------------------------------
        // private elements can only be accessed from functions defined within
        // the class structure
        //------------------------------------------------------------------
        int privateMember; // integer
        void privateMethod1(int arg1, int arg2); //method with arguments and no return
        int privateMethod2(); //method with no arguments and return type int
protected:
        // --------------Protected Elements---------------------------------
        // protected elements can only be accessed from functions defined within
        // the class structure and classes that are friends of this class
        // which can be declared as
        //--------------------------------------------------------------------
        friend class FriendlyClass;
        // now FriendlyClass can access protected elements of ClassName
        int protectedMember1;
        void protectedMethod1(); // Function Prototype is defined in "ClassName.cpp"
        void protectedMethod2()
        {
                // Method defined within the class structure
                // both direct definition and function prototypes have pros and cons

                // do stuff...
        }
public:
        // --------------Public Elements----------------------
        // public elements can be accessed by anyone
        int publicMember;

        // Constructors/ Destructors and operator overload are declared here
        // …… continued in next page
```

```
        //--------Constructors and Destructors----------------
        // default constructor with no input arguments
        //-----------------------------------------------------
        ClassName()
        {
                //Initialize members and any inherited classes
        }

        //Constructor with inputs and initialization
        ClassName(int arg1, float arg2) :
                privateMember(arg1),
                ParentClass(arg1, arg2) // initialize parent classes if needed
        {
                //Do more stuff
        }
        //Constructor prototype
        ClassName(int arg1); // Define later

        //Copy Constructor
        ClassName(const ClassName&)
        {
                // Do stuff to copy things from input Class to this class
        }

        //Destructor
        ~ClassName()
        {
                //how to destroy elements of this object to free up memory when
                //an object is deleted
        }

        // --------------Operator Overload--------------
        // operators are (+, -, *, (), [], <<, >> etc.)
        // Operators separate two objects for example,
        //    C = A+B
        // operator + separates A and B, and operator =
        // separates the result of (A+B) and C.

        // the "this" pointer is assigned to the object
        // on the left of the operator
        //  ---------------------------------------------

        //Output stream operator
        friend std::ostream& operator <<( std::ostream & os, const ClassName &A);
        //Assignment Operator
        ClassName& operator= (const ClassName& D);
        //Addition Operator
        ClassName operator+ (const ClassName& B);
        //-------------Other Public Members and Methods-------------------------

};
}
#endif
```

```cpp
// This is the implementation file (AKA ClasName.cpp )
#include "ClassName.h"
//--------Use this file to define your class attributes ----------------
namespace myNamespace {

	void ClassName::privateMethod1(int arg1, int arg2)
	{
		//Do stuff

		return;
	}
	int ClassName::privateMethod2()
	{
		//Do stuff

		return 0;
	}
	void ClassName::protectedMethod1 ()
	{
		//Do stuff

		return;
	}
	std::ostream& operator <<( std::ostream& os, const ClassName &A)
	{
		os << "print to out stream" << std::endl;
		return os;
	}
	ClassName& ClassName::operator= (const ClassName& D)
	{
		// this returns the reference to the class object
		// on the left of the equal sign
		// so for C = D, this->elements would access the
		// elements of C

		// do stuff to assign attributes to this pointer
		return *this;
	}

	//Addition Operator
	ClassName ClassName::operator+ (const ClassName& B)
	{
		// returns an entirely new object for the example
		// C = A+B.
		// We manipulate properties of C using the pointer "this", which
		// points to A and can access the elements of A, and input B. Then
		// we return the object C.
		ClassName C();
		C.publicMember = this->privateMember + B.publicMember;
		return D;
	}

}
```

3 References & Resources

3.1 References

The following references may be helpful to complete the lab.

1. https://en.cppreference.com/w/cpp/language/operators

4 Pre-lab

This section should be worked on prior to arriving in lab.

4.1 Items needed to complete lab two

Be sure to have the exercises from lab one ready as you will use the functions you created to populate the code in this lab.

5 Laboratory

Complete the exercises and save all code. Combine your solutions and code from lab one with the material you will create in this lab. Follow the procedure to turn in your work in the following procedure.

5.1 Requirements

Bring your raspberry pi (RPI) so that you can develop your code. Be sure to have lab one's functions created.

5.2 Procedure

1. Assuming you named the files, which hold your functions from lab one, *SPmath.h* and *SPmath.cpp*, create two more files which will contain the information about the matrix class. I named mine *SPMatrix.h* and *SPMatrix.cpp*.
2. Make sure to include *SPmath.h* in *SPMatrix.h*.to aid in creating your matrix class.
3. Create one more file named *labtwo.cpp* which will be used to test out your matrix operations.
4. Define your main function in *labtwo.cpp*.
5. You can test out your code using my example main function and the terminal printout of its results. If you get the same answers then you can be fairly confident your code is correct.
6. When finished with the exercises in the next section, combine the solutions from lab one and lab two into one submission on Carmen. The submission will be in the form of a zip file and must conform to the following requirements:
    a. The zip file should be named as follows
        i. SP19_lab01_Lastname_dotnumber.zip
    b. Inside the zip files should be only the following
        i. Six programming files which are (what you call them doesn't matter, just make sure I can tell what each file does)
            1. *labone.cpp* – main implementation file for lab one
            2. *labtwo.cpp* – main implementation file for lab two
            3. *SPmath.h* – header file for lab one
            4. *SPmath.cpp* – implementation file for lab one

5. *SPMatrix.h* – header file for lab two
6. *SPMatrix.cpp* – implementation file for lab two.

ii. One report in PDF format which will contain only the following in 3 pages or less,
    1. Date, Name, Class, and Lab number(s) at the top.
    2. Section one:
        a. Describe which data structures you used to hold your information in lab one.
        b. Very, very briefly describe any function you created other than the ones required for lab one.
        c. Briefly describe how you implemented the functions created in lab one to efficiently develop your matrix class in lab two.
    3. Section two
        a. Describe the algorithm you developed to solve the matrix inversion problem in lab one either in words or pseudo code.
        b. Be sure to say which numerical technique you used (i.e. Gauss-Elimination, Cofactor method, etc.).
        c. Describe how you checked to see if the matrix was invertible or not.
    4. Section three
        a. What information do you think would have helped you better understand and complete the exercises in this lab?
        b. Include picture or two of the terminal printouts from the main implementation files created in lab one and lab two, displaying your results.

5.3 Exercises

Create a matrix class (I called my class **MatF,** name yours whatever you'd like) which satisfies the following requirements

1. The class contains the ability to perform matrix multiplication, addition, subtraction, transposition, and inversion operations.
2. The class should contain protected members which represent
    a. The structure where the matrix data is stored
    b. The dimensions of the matrix (i.e. number of rows and number of columns)
    c. A Boolean that is true if the data structure has been initialized and false otherwise.
3. The class should overload the following operators (for help see the class template above and example implementations below).
    a. **operator<<**
        i. This will print the matrix's data to the output stream.
        ii. `std::cout << A;`
    b. **operator=**
        i. This will assign the information contained in the matrix on the right-hand side (RHS) of the equal sign, B, to the matrix on the left-hand side (LHS), A.
        ii. `A = B;`
    c. **operator+**
        i. This will create a new matrix, C, add the data from the matrix on the RHS (B) with the data from the matrix on the LHS (A), assign the results to matrix C, and return matrix C.
        ii. `C = A + B;`
            1. Remember the pointer **this->** accesses the methods and members from objects on the left-hand side of the operator (see examples above)
    d. **operator-**
        i. This will create a new matrix, C, subtract the data from the matrix on the RHS (B) from the data from the matrix on the LHS (A), assign the results to matrix C, and return matrix C.
        ii. `C = A - B;`
    e. **operator***
        i. This will create a new matrix, C, multiply the data from the matrix on the RHS (B) with the data from the matrix on the LHS (A), assign the results to matrix C, and return matrix C. Order of operations matters here!
        ii. `C = A * B;`
4. The class should also include methods for generating the transpose of the matrix, inverse of the matrix, and three constructors which can
    a. Take no inputs → **MatF()**
    b. Take three inputs → **MatF(int rows, int cols, float val)**

i. This would assign the initiated object's members the information about the number of rows and columns and initialize its matrix's data to be a matrix of size (rows by cols) filled with the value dictated by *float val*.

c. Take one input of type **MatF_t** (or a matrix float type) → `MatF(MatF_t data)`
   i. This would extract the needed information from the data and assign it to the initiated object's members.

Demonstrate that your code is working using the main file code as shown in Figure 2. Compare your results to the solutions presented in Figure 3. I will have a similar test script that will test out your code and you will be graded on its performance, as well as other factors.

6 Rubric

Exercise	*Performance*	*Completeness*	*Cleanliness*	*Clarity*	*Total*
Lab One: Code	/ 50	/ 30	/ 10	/ 10	/ 100
Lab Two: Code	/ 50	/ 30	/ 10	/ 10	/ 100
Report	-	/ 50	-	-	/ 50
Total	**/ 100**	**/ 110**	**/20**	**/20**	**/250**

1. **Performance**– Does your code produce the correct results.
2. **Completeness**– How much of the exercise did you complete.
3. **Cleanliness** – Of the completed portion of the exercise how well organized and structured is the work (i.e. is your code compact and clean, does it flow from one section to another)
4. **Clarity –** Of the completed portion of the exercise, how easy is it to infer what is occurring from the context of your work (i.e. do you use concise and proper use of commenting, if needed, did you explain your work/methodology in your report).

```
Example Solution for Lab One
#Terminal print out for example of matrix inversion solution
Dm =
5       3.3     2.34
6.7     7.32    9.3
8.123   7.345   22.34

Inverse Procedure: inv(Dm):
----------------
Augmented matrix
----------------
5       3.3     2.34    1       0       0
6.7     7.32    9.3     0       1       0
8.123   7.345   22.34   0       0       1
----------------
new row
----------------
row operation
1       0.66    0.468   0.2     0       0
6.7     7.32    9.3     0       1       0
8.123   7.345   22.34   0       0       1

row operation
1       0.66    0.468   0.2     0       0
0       2.898   6.1644  -1.34   1       0
8.123   7.345   22.34   0       0       1

row operation
1       0.66    0.468   0.2     0       0
0       2.898   6.1644  -1.34   1       0
0       1.98382 18.5384 -1.6246 0       1
----------------
new row
----------------
row operation
1       0.66    0.468   0.2     0       0
0       1       2.12712 -0.462388       0.345065        0
0       1.98382 18.5384 -1.6246 0       1

row operation
1       0       -0.9359 0.505176        -0.227743       0
0       1       2.12712 -0.462388       0.345065        0
0       1.98382 18.5384 -1.6246 0       1

row operation
1       0       -0.9359 0.505176        -0.227743       0
0       1       2.12712 -0.462388       0.345065        0
0       0       14.3186 -0.707306       -0.684548       1
----------------
new row
----------------
row operation
1       0       -0.9359 0.505176        -0.227743       0
0       1       2.12712 -0.462388       0.345065        0
0       0       1       -0.0493977      -0.0478083      0.0698392

row operation
1       0       0       0.458945        -0.272487       0.0653625
0       1       2.12712 -0.462388       0.345065        0
0       0       1       -0.0493977      -0.0478083      0.0698392

row operation
1       0       0       0.458945        -0.272487       0.0653625
0       1       0       -0.357313       0.446759        -0.148556
0       0       1       -0.0493977      -0.0478083      0.0698392
----------------
Done
----------------

inv(Dm)=
0.458945        -0.272487       0.0653625
-0.357313       0.446759        -0.148556
-0.0493977      -0.0478083      0.0698392
```

Figure 1 – Hints for solving matrix inversion

```cpp
// Example main implementation code for lab two
#include<iostream>
#include"SPMatrix.h"

int main()
{
	/*
	Main Code
	*/
	sp::MatF_t data = { { 5.0f, 3.3f, 2.34f },{ 6.7f, 7.32f, 9.3f },{ 8.123f, 7.345f, 22.34f } };
	sp::MatF A(data);
	std::cout << "Matrix A = "<<std::endl << A << std::endl;
	sp::MatF Ainv = A.inv();
	std::cout << "Matrix Ainv = inverse(A) " << std::endl << Ainv << std::endl;
	sp::MatF B = A * Ainv;
	std::cout << "Matrix B = A * Ainv " << std::endl << B << std::endl;
	sp::MatF C = ((A.t() * Ainv + A - B) * A).inv();
	std::cout << "Matrix C = inverse((transpose(A) * Ainv + A - B) * A)" << std::endl << C << std::endl;
}
```

Figure 2 – Example main file "labtwo.cpp" for executing the matrix class operations

```
Comparative Solution for Lab Two
#Terminal print out for correct test solution for lab two
Matrix A =
5        3.3      2.34
6.7      7.32     9.3
8.123    7.345    22.34


Matrix Ainv = inverse(A)
 0.458945        -0.272487        0.0653625
-0.357313         0.446759       -0.148556
-0.0493977       -0.0478083       0.0698392


Matrix B = A * Ainv
1                 4.47035e-08      1.49012e-08
-8.9407e-08       1                5.96046e-08
0                -1.19209e-07      1


Matrix C = inverse((transpose(A) * Ainv + A - B) * A)
 2.08895         -1.78587         0.524027
-2.21635          1.93978        -0.578206
-0.0137429       -0.00717579      0.00760291
```

Figure 3 – Terminal printout of lab two solution comparison