

CSCE 221 Cover Page
Programming Assignment #1
Due by February 4 midnight to eCampus

First Name: Asa Last Name: Hayes UIN: 525003952

User Name AsaHayes E-mail address: asahayes@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more: [Aggie Honor System Office](#)

Type of sources	
People	Josiah Egner
Web pages (provide URL)	Clarification on use of templated functions.
Other Sources	Several programming assignments from CSCE-121 with Dr. Moore

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

“On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.”

Your Name: Asa R. Hayes Date: 7 February, 2018

Report

February 7, 2018

Part I

Program Description and Purpose of Assignment

You want to implement a data structure that allows to tabulate data coming from many different models such as communication or social networks. The entries in a table may express a relation between two groups of people, e.g. the number 1 could denote friends and 0 otherwise. This type of data structure could be also used to represent a location of an object in a two-dimensional space using its coordinates, e.g. a pair (i, j) to refer to a particular element of the table. These tables are called two-dimensional arrays or matrices. The Background section of this assignment provides some basic information about matrices and their operations.

The purpose of this individual programming assignment is to learn about an elementary design, implementation, and testing of a simple C++ class called `My_matrix`. The class implementation allows you to understand and overview the basic C++ concepts like pointers, memory allocation, deallocation, dynamic arrays, constructors, copy or move constructors and assignments, destructors, operator overloading, reading from and writing to a file.

Part II

Description of Data Structures

In the first phase of the assignment, implement in C++ a class `My_matrix` that can hold data of integer type (`int`). The two parameters representing a matrix dimensions are usually not known in advance so it is necessary to allocate the arrays dynamically. In the second phase, you will need to write a generic version of the class `My_matrix` that can handle different types of numerical data.

The `My_matrix` class takes either a set of two dimensions (for rows and columns) or another `My_matrix` object as input. From this, the object creates an `int**` (or `T**` in part 2), which is an array containing pointers to the sub-arrays that make up the matrix part of the `My_matrix` object. Once the matrix is created, it can be filled with data from a file, data from another `My_matrix` object, or manually set per element. Input and display are facilitated by the overloading of the input and output operators. Operators `*` and `+` are also overloaded, to allow for some commonly used matrix operations for compatible matrices.

Part III

Instructions for Compilation

No additional effort is needed for compilation, the makefile is configured for each part to compile entirely with “make all” and run with “./main”. The only caution would be to not remove any of the test input or output files, as the testing part of `main{}` requires them.

Part IV

Logical Exceptions; Bug Descriptions

Apart from the code blocks that lead to errors intentionally, there do not seem to be any places for logical errors or bugs besides the plain invalid input errors.

Part V

C++ Object Oriented/Generic Programming Features

The most important features of object-oriented programming that are shown in this project are reusability and modularity. As `My_matrix` has a large amount of usages, having to manually create a structure each and every time one was needed would ruin readability and maintainability. Being able to simply create new `My_matrix` objects with one command not only creates better looking code, but reduces debug time needed, due to less variables having to be dragged around. The modularity of the `My_matrix` class is also important,

as most of the functions contained within have no analogue within standard libraries and need to be tailored to the function. For example, the multiplication operator is much differently applied to a matrix than just 2 numbers as it would usually be interpreted, but allowing a specialized multiplier, we conserve the idea of the operator while still having it carry out the proper function, instead of creating an entirely new operator. As for general programming, the use of generic type T allows for much more versatility of what My_matrix is able to hold, allowing all types of numbers instead of just the default integers.

Part VI

Evidences of Testing

Note: The tests in the screenshots were run on Windows 7 with MinGW, but results were verified and identical when run on the TAMU UNIX server.

1 Evidence: Part 1

1.1 Test 1: Initialization of My_matrix objects, setting of matrix elements, demonstration of output operator

```

22  /**
23  //Initialize new My_matrix and verifies dimensions
24  cout << endl << "Test 1: " << endl;
25  My_matrix m1(2,2);
26  cout << "m1 created" << endl;
27  cout << "m1 - Rows: " << m1.number_of_rows() << endl;
28  cout << "m1 - Columns: " << m1.number_of_columns() << endl << endl;
29
30  //assigns and demonstrates proper setting and getting of elements in My_matrix
31  cout << "Manual setting + output with elem(i,j)" << endl;
32  m1.elem(0,0) = 1;
33  cout << m1.elem(0,0) << " ";
34  m1.elem(0,1) = 2;
35  cout << m1.elem(0,1) << endl;
36  m1.elem(1,0) = 3;
37  cout << m1.elem(1,0) << " ";
38  m1.elem(1,1) = 4;
39  cout << m1.elem(1,1) << endl;
40
41  cout << "Overloaded << output for m1" << endl << m1 << endl;
42
43  // Usually m1 is deleted, but it will be used in test 3.
44  //delete m1;
45  /**
46
47  // Test 2
48  // * Create an object of type My_matrix, called m2, using (default)
49  // constructor

```

```

C:\Windows\system32\cmd.exe
c++ -std=c++11 -c -g main.cpp
c++ -std=c++11 My_matrix.o main.o -o main

Test 1:
m1 created
m1 - Rows: 2
m1 - Columns: 2

Manual setting + output with elem(i,j)
1 2
3 4
Overloaded << output for m1
1 2
3 4

Test 2:
Data sent to output file:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4

Test 3:
m3 copy constructed from m1

```

1.2 Test 2: Reading from input file into new My_matrix, displaying, reading to output file

```

58 //*/
59 cout << "Test 2: " << endl;
60
61 ifstream infs("test_input.txt");
62 ofstream outfs("test_output.txt");
63 if (!infs.is_open()) {
64     cerr << "Failed to open in file: " << endl;
65     return 1;
66 }
67 if (!outfs.is_open()) {
68     cerr << "Failed to open out file: " << endl;
69     return 1;
70 }
71
72 int temp = 0, ntemp = 0, mtemp = 0;
73 for (int i = 0; i < 2; ++i) {
74     if (!infs >> temp) {
75         switch (i) {
76             case 0: ntemp = temp;
77                 break;
78             case 1: mtemp = temp;
79                 break;
80         }
81     } else {
82         cout << "Error: not enough data." << endl;
83         return 1;
84     }
85 }
86
87 My_matrix m2(ntemp, mtemp);
88
89 infs >> m2;
90
91 outfs << m2.number_of_rows() << " " << m2.number_of_columns() << endl << m2;
92
93 cout << endl << "Data sent to output file: " << endl << m2 << endl;
94 //*/

```

```

1 2
3 4

Test 2:
Data sent to output file:
1 1 1 1
2 2 2 2
3 3 2 3
4 4 4 4

Test 3:
m3 copy constructed from m1
m3:
1 2
3 4

```

1.3 Test 3: Copy Constructor and Copy Assignment Operators

```

94 //*/
95
96 // Test 3
97 // * Use the copy constructor to make a copy of m1 called m3
98 // * Apply the copy assignment to m1 called m4
99 // * Display m3 and m4 on the screen using the operator <<
100
101 cout << "Test 3: " << endl;
102
103 cout << "m3 copy constructed from m1" << endl;
104 My_matrix m3(m1);
105 cout << "m3:" << endl << m3 << endl;
106 cout << "m1:" << endl << m1 << endl;
107 cout << "m4 copy assigned from m3" << endl;
108 My_matrix m4;
109 m4 = m3;
110 cout << "m4:" << endl << m4 << endl;
111
112 // Test 4
113 // * Test the matrix multiplication operator (operator*)
114 // * Apply the multiplication operator to valid and invalid cases
115 // * Display the resulting matrix and its number of rows and columns
116 //*/
117 cout << "Test 4: " << endl;
118
119 ifstream infs 4a("test_input1.txt");
120 ifstream infs 4b("test_input2.txt");
121 if (!infs 4a.is_open()) {

```

```

4 4 4 4

Test 3:
m3 copy constructed from m1
m3:
1 2
3 4

m1:
1 2
3 4

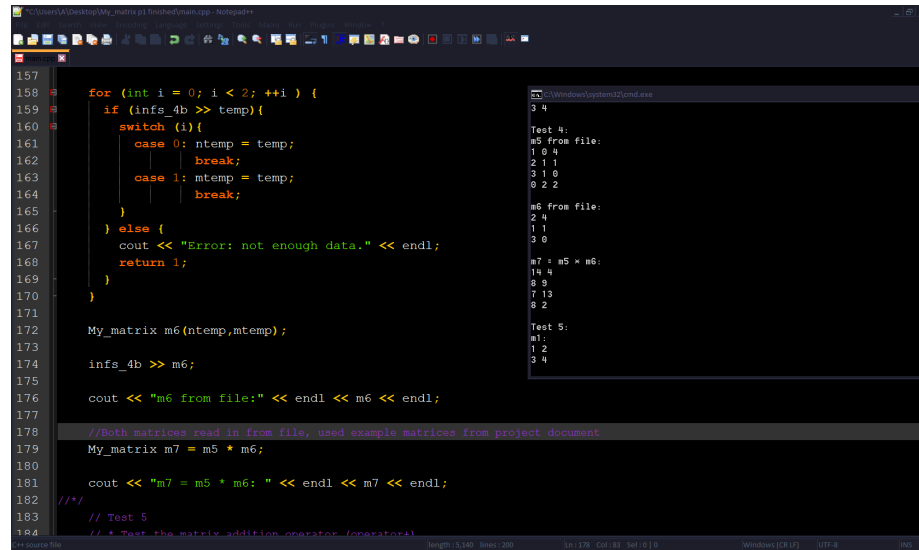
m4 copy assigned from m3
m4:
1 2
3 4

Test 4:
m5 from file:
1 0 4
2 1 1
3 1 0
0 2 2

m6 from file:

```

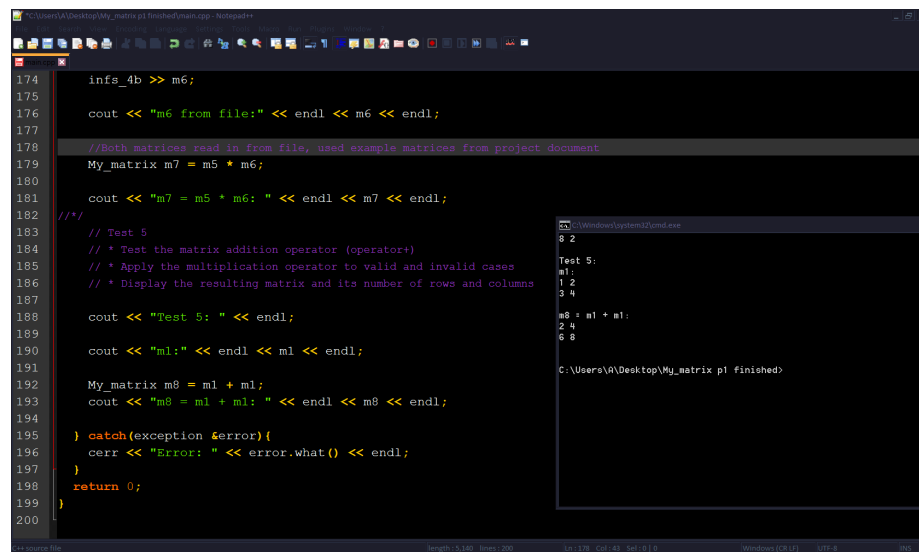
1.4 Test 4: Multiplication Operator



```
157
158
159     for (int i = 0; i < 2; ++i ) {
160         if (infs_4b >> temp){
161             switch (i){
162                 case 0: ntemp = temp;
163                     break;
164                 case 1: mtemp = temp;
165                     break;
166             }
167         } else {
168             cout << "Error: not enough data." << endl;
169             return i;
170         }
171     }
172
173     My_matrix m6(ntemp,mtemp);
174
175     infs_4b >> m6;
176
177     cout << "m6 from file:" << endl << m6 << endl;
178
179     //Both matrices read in from file, used example matrices from project document
180     My_matrix m7 = m5 * m6;
181
182     cout << "m7 = m5 * m6: " << endl << m7 << endl;
183
184     /**
185     // Test 5
186     // * Test the matrix addition operator (operator+)
187     // * Apply the multiplication operator to valid and invalid cases
188     // * Display the resulting matrix and its number of rows and columns
189     */
190
191     cout << "Test 5: " << endl;
192
193     cout << "m1:" << endl << m1 << endl;
194
195     My_matrix m8 = m1 + m1;
196     cout << "m8 = m1 + m1: " << endl << m8 << endl;
197
198     } catch(exception &error){
199         cerr << "Error: " << error.what() << endl;
200     }
201
202     return 0;
203 }
```

```
3 4
Test 4:
m5 from file:
1 0 4
2 1 1
3 1 0
0 2 2
m6 from file:
2 4
1 1
3 0
m7 = m5 * m6:
14 4
0 9
7 18
0 2
Test 5:
m1:
1 2
3 4
```

1.5 Test 5: Addition Operator



```
174
175
176     cout << "m6 from file:" << endl << m6 << endl;
177
178     //Both matrices read in from file, used example matrices from project document
179     My_matrix m7 = m5 * m6;
180
181     cout << "m7 = m5 * m6: " << endl << m7 << endl;
182
183     /**
184     // Test 5
185     // * Test the matrix addition operator (operator+)
186     // * Apply the multiplication operator to valid and invalid cases
187     // * Display the resulting matrix and its number of rows and columns
188     */
189
190     cout << "Test 5: " << endl;
191
192     cout << "m1:" << endl << m1 << endl;
193
194     My_matrix m8 = m1 + m1;
195     cout << "m8 = m1 + m1: " << endl << m8 << endl;
196
197     } catch(exception &error){
198         cerr << "Error: " << error.what() << endl;
199     }
200
201     return 0;
202 }
```

```
0 2
Test 5:
m1:
1 2
3 4
m8 = m1 + m1:
2 4
6 8
C:\Users\A\Desktop\My_matrix p1 finished
```

2 Evidence: Part 2

Screenshots will be focusing on the `My_matrix<double>` type, as the `My_matrix<long>` type is visibly identical to `My_matrix<int>` which was demonstrated in part 1.

2.1 Test 1: Initialization of My_matrix<double> object, setting of matrix elements, demonstration of output operator

```

22 //Initialize new My_matrix and define dimensions
23 cout << endl << "Test 1: " << endl;
24
25 My_matrix<int> m1(2,2);
26 cout << "m1 created, type <int>" << endl;
27 cout << "m1 - Rows: " << m1.number_of_rows() << endl;
28 cout << "m1 - Columns: " << m1.number_of_columns() << endl << endl;
29
30 My_matrix<double> m1a(2,2);
31 cout << "m1a created, type <double>" << endl;
32 cout << "m1a - Rows: " << m1a.number_of_rows() << endl;
33 cout << "m1a - Columns: " << m1a.number_of_columns() << endl << endl;
34
35 My_matrix<long> m1b(2,2);
36 cout << "m1b created, type <long>" << endl;
37 cout << "m1b - Rows: " << m1b.number_of_rows() << endl;
38 cout << "m1b - Columns: " << m1b.number_of_columns() << endl << endl;
39
40 //Assigns and demonstrates proper setting and getting of elements in My_matrix
41 cout << "Manual setting of m1 - output with elem(i,j)" << endl;
42 m1.elem(0,0) = 1;
43 cout << m1.elem(0,0) << " ";
44 m1.elem(0,1) = 2;
45 cout << m1.elem(0,1) << endl;
46 m1.elem(1,0) = 3;
47 cout << m1.elem(1,0) << " ";
48 m1.elem(1,1) = 4;
49 cout << m1.elem(1,1) << endl;
50
51 cout << "Manual setting of m1a + output with elem(i,j)" << endl;
52 m1a.elem(0,0) = 1.1;
53 cout << m1a.elem(0,0) << " ";
54 m1a.elem(0,1) = 2.2;
55 cout << m1a.elem(0,1) << endl;
56 m1a.elem(1,0) = 3.3;
57 cout << m1a.elem(1,0) << " ";
58 m1a.elem(1,1) = 4.4;
59 cout << m1a.elem(1,1) << endl;

```

```

c++ -std=c++11 main.o -o main

Test 1:
m1 created, type <int>
m1 - Rows: 2
m1 - Columns: 2

m1a created, type <double>
m1a - Rows: 2
m1a - Columns: 2

m1b created, type <long>
m1b - Rows: 2
m1b - Columns: 2

Manual setting of m1 + output with elem(i,j)
1 2
3 4
Manual setting of m1a + output with elem(i,j)
1.1 2.2
3.3 4.4
Manual setting of m1b + output with elem(i,j)
1 2
3 4

Overloaded << output for m1
1 2
3 4

Overloaded << output for m1b
1 2
3 4

Overloaded << output for m1c
1 2
3 4

```

2.2 Test 2: Reading from input file into new My_matrix<double>, displaying, reading to output file

```

130 // double >> and << test
131
132 ifstream infsa("test_input.txt");
133 ofstream outfsa("test_output.txt");
134 if (!infsa.is_open()) {
135     cerr << "Failed to open in file: " << endl;
136     return 1;
137 }
138 if (!outfsa.is_open()) {
139     cerr << "Failed to open out file: " << endl;
140     return 1;
141 }
142
143 for (int i = 0; i < 2; ++i) {
144     if (infsa >> temp) {
145         switch (i) {
146             case 0: ntemp = temp;
147                     break;
148             case 1: mtemp = temp;
149                     break;
150         }
151     } else {
152         cout << "Error: not enough data." << endl;
153         return 1;
154     }
155 }
156
157 My_matrix<double> m2a(ntemp, mtemp);
158
159 infsa >> m2a;
160
161 outfsa << m2a.number_of_rows() << " " << m2a.number_of_columns() << endl << m2a;
162 cout << endl << "Data sent to output file: " << endl << m2a << endl;
163

```

```

Test 2:

Data sent to output file:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4

Data sent to output file:
1.1 1.1 1.1 1.1
2.2 2.2 2.2 2.2
3.3 3.3 3.3 3.3
4.4 4.4 4.4 4.4

Data sent to output file:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4

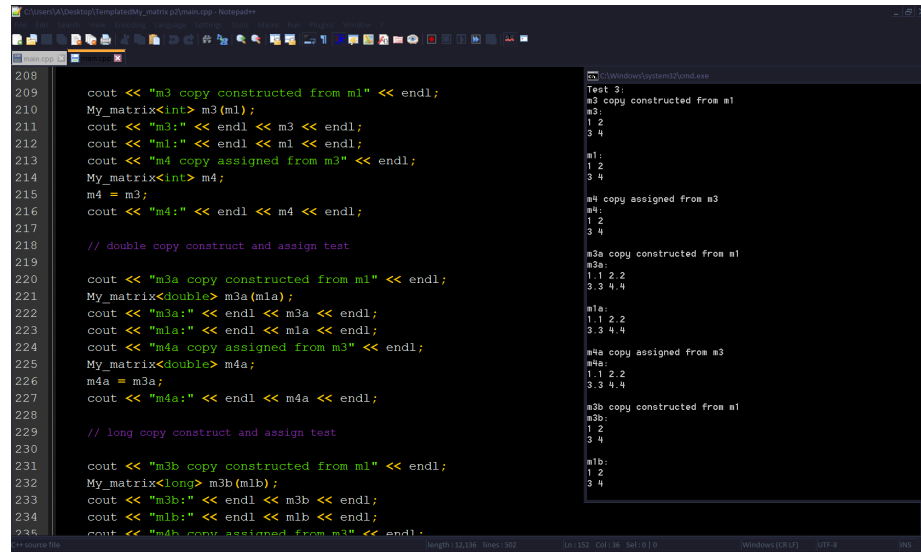
Test 3:
m3 copy constructed from m1
m3:
1 2
3 4

m1:
1 2
3 4

m4 copy assigned from m3
m4:
1 2
3 4

```

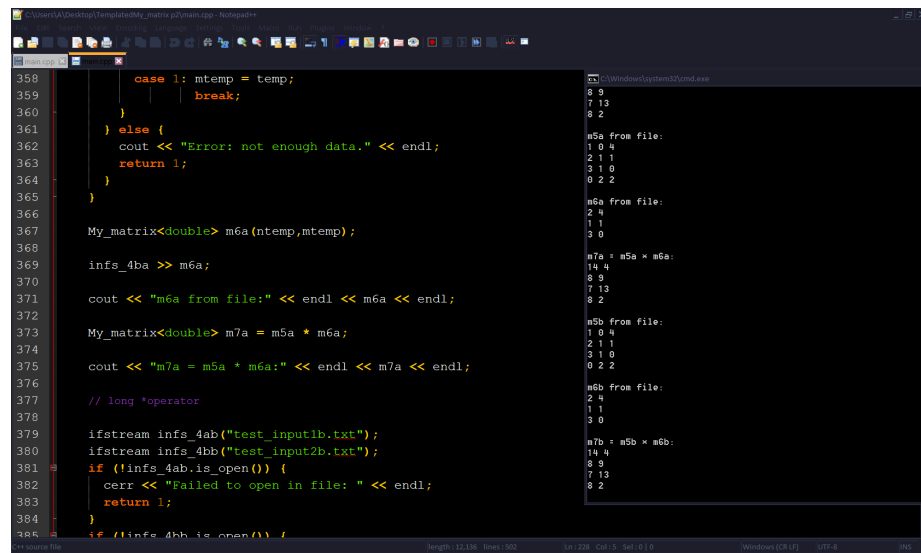
2.3 Test 3: Copy Constructor and Copy Assignment Operators



```
208 cout << "m3 copy constructed from m1" << endl;
209 My_matrix<int> m3(m1);
210 cout << "m3:" << endl << m3 << endl;
211 cout << "m1:" << endl << m1 << endl;
212 cout << "m4 copy assigned from m3" << endl;
213 My_matrix<int> m4;
214 m4 = m3;
215 cout << "m4:" << endl << m4 << endl;
216
217 // double copy construct and assign test
218
219 cout << "m3a copy constructed from m1" << endl;
220 My_matrix<double> m3a(m1a);
221 cout << "m3a:" << endl << m3a << endl;
222 cout << "m1a:" << endl << m1a << endl;
223 cout << "m4a copy assigned from m3" << endl;
224 My_matrix<double> m4a;
225 m4a = m3a;
226 cout << "m4a:" << endl << m4a << endl;
227
228 // long copy construct and assign test
229
230 cout << "m3b copy constructed from m1" << endl;
231 My_matrix<long> m3b(m1b);
232 cout << "m3b:" << endl << m3b << endl;
233 cout << "m1b:" << endl << m1b << endl;
234 cout << "m4b copy assigned from m3" << endl;
```

```
Test 3:
m3 copy constructed from m1
m3:
1 2
3 4
m1:
1 2
3 4
m4 copy assigned from m3
m4:
1 2
3 4
m3a copy constructed from m1
m3a:
1 1 2 2
3 3 4 4
m1a:
1 1 2 2
3 3 4 4
m4a copy assigned from m3
m4a:
1 1 2 2
3 3 4 4
m3b copy constructed from m1
m3b:
1 2
3 4
m1b:
1 2
3 4
```

2.4 Test 4: Multiplication Operator, with test for incompatible matrices



```
358 case 1: mtemp = temp;
359 break;
360 }
361 } else {
362 cout << "Error: not enough data." << endl;
363 return 1;
364 }
365 }
366
367 My_matrix<double> m6a(ntemp,mtemp);
368
369 ifs_4ba >> m6a;
370
371 cout << "m6a from file:" << endl << m6a << endl;
372
373 My_matrix<double> m7a = m5a * m6a;
374
375 cout << "m7a = m5a * m6a:" << endl << m7a << endl;
376
377 // long *operator
378
379 ifstream ifs_4ab("test_input1b.txt");
380 ifstream ifs_4bb("test_input2b.txt");
381 if (!ifs_4ab.is_open()) {
382 cerr << "Failed to open in file: " << endl;
383 return 1;
384 }
385 if (!ifs_4bb.is_open()) {
```

```
m5a from file:
1 0 4
2 1 1
3 1 0
0 2 2
m6a from file:
2 4
1 1
3 0
m7a = m5a * m6a:
14 4
8 9
7 13
8 2
m5b from file:
1 0 4
2 1 1
3 1 0
0 2 2
m6b from file:
2 4
1 1
3 0
m7b = m5b * m6b:
14 4
8 9
7 13
8 2
```



```

433     ifs_4bb >> m6b;
434
435     cout << "m6b from file:" << endl << m6b << endl;
436
437     My_matrix<long> m7b = m5b * m6b;
438
439     cout << "m7b = m5b * m6b:" << endl << m7b << endl;
440
441     // incompatible matrices for *operator test, should give error as 6a is not a square matrix.
442     try{
443         cout << "m6:" << endl << m6 << endl;
444
445         cout << "m10 = m6 + m6:" << endl;
446         My_matrix<int> m10 = m6 * m6;
447         cout << m10 << endl;
448     } catch(exception &error){
449         cerr << "Error: " << error.what() << endl;
450     }
451
452     // Test 5
453     // * Test the matrix addition operator (operator+)
454     // * Apply the multiplication operator to valid and invalid cases
455     // * Display the resulting matrix and its number of rows and columns
456     cout << endl << "Test 5: " << endl;
457
458
459
460

```

```

8 2
m6:
2 4
1 1
3 0
m10 = m6 * m6:
Error: Incompatible matrices

```

2.5 Test 5: Addition Operator, with test for incompatible matrices

```

469     // double +operator test
470
471     cout << "m1a:" << endl << m1 << endl;
472
473     My_matrix<double> m8a = m1a + m1a;
474     cout << "m8a = m1a + m1a: " << endl << m8a << endl;
475
476     // long +operator test
477
478     cout << "m1b:" << endl << m1b << endl;
479
480     My_matrix<long> m8b = m1b + m1b;
481     cout << "m8b = m1b + m1b: " << endl << m8b << endl;
482
483     // Differently sized matrices, should give error
484     try{
485         cout << "m1:" << endl << m1 << endl;
486         cout << "m2:" << endl << m2 << endl;
487
488         cout << "m9 = m1 + m2: " << endl;
489         My_matrix<int> m9 = m1 + m2;
490         cout << m9 << endl;
491     } catch(exception &error){
492         cerr << "Error: " << error.what() << endl;
493     }
494
495

```

```

m1a:
1 2
3 4
m8a = m1a + m1a:
2 2 4 4
6 6 8 8
m1b:
1 2
3 4
m8b = m1b + m1b:
2 4
6 8
m1:
1 2
3 4
m2:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
m9 = m1 + m2:
Error: Incompatible matrices

```