ENSIA 2021-2022
Object–Oriented Programming
Lab 8: On Chapters Exception Handling and Class Templates

Exercise 1

Different exceptions can arise when doing mathematical calculations. We want you to define various classes to capture and handle such exceptions.

1. Define class **MathException,** derived from class **exception,** and which has a protected string data member *message*, a constructor and a function *what()* which returns the value of message.
2. Define three classes **DivideByZeroException**, **OverflowException**, and **RootOfNegativeException**. Each of these classes is derived from class **MathException** and has a constructor which passes the appropriate message to the **MathException** constructor. An object of the first class is thrown when a division by zero is about to occur on two integer values entered by the user; an object of the second class is thrown when (just for the purposes of this exercise) a very big number (say >5000000) is read; and an object of the third class is thrown when the calculation of the root of a negative number entered by the user is about to be calculated. Otherwise, the operations are performed normally. (If no OverflowException, you can just output the number just read.)
3. Write a driver which, in a try block, does the necessary to throw the appropriate exception each time. Subsequent catch blocks should match the exceptions and handle them by writing the proper message.

Problem 2

1. create a class **DoubleSubscriptedArray** that has similar features to class Array in Figs. 11.6-11.7. At construction time, the class should be able to create an array of any number of rows and any number of columns. The class should supply operator() to perform double-subscripting operations. For example, in a 3-by-5 **DoubleSubscriptedArray** called a, the user could write a(1, 3) to access the element at row 1 and column 3. The underlying representation of the double-subscripted array should be a single-subscripted array of integers with *rows * columns* number of elements. Function *operator()* should perform the proper pointer arithmetic to access each element of the array. There should be two versions of operator() -- one that returns **int &** (so that an element of a **DoubleSubscriptedArray** can be used as an lvalue) and one that returns **const int &** (so that an element of a **const DoubleSubscriptedArray** can be used only as an rvalue). The class should also provide the following operators: ==, !=, =, << (for outputting the array in row and column format) and >> (for inputting the entire array contents).

   By default, the array has 10 * 10 elements.
2. Write a driver to test the various member functions; the following is a sample output:

Sample Output:

```
Uninitialized array “a” is:
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       0       0       0       0       0       0
Uninitialized array “b” is:
0       0
0       0
0       0
0       0
0       0
0       0
0       0
0       0
Initialized array “a” is now:
89      13      54      19      27      30      76
96      22      21      7       97      26      43
64      90      4       86      6       66      92
92      86      19      80      76      60      49
84      15      56      5       39      97      53
12      34      49      91      54      98      39
Assigning b = a :
89      13      54      19      27      30      76
96      22      21      7       97      26      43
64      90      4       86      6       66      92
92      86      19      80      76      60      49
84      15      56      5       39      97      53
12      34      49      91      54      98      39
“a” was found to be equal to “b”
The element (2, 1) of array “a” is: 90
Changed element (2, 1) to -1 :
89      13      54      19      27      30      76
96      22      21      7       97      26      43
64      -1      4       86      6       66      92
92      86      19      80      76      60      49
84      15      56      5       39      97      53
12      34      49      91      54      98      39
“a” was found not to be equal to “b”
```

3. Create a template of the **DoubleSubscriptedArray** using a type parameter **elementType** when defining this class. Name your template class **Table**. This template enables Table objects to be instantiated with a specified element type at compile time. The underlying representantion of the double-subscripted array should be a single-subscripted array with *rows * columns* number of elements.
4. Create a driver program to test the capabilities of your class. The following is a sample output:

Sample Output:

```
Uninitialized array “a” is:
0       0       0       0       0       0       0
```

```
0       0       0       0       0       0       0
0       0       0       0       0       0       0
0       0       0       0       0       0       0
```

Uninitialized array “b” is:

Initialized array “a” is now:

```
89      13      54      19      27      30      76
96      22      21      7       97      26      43
64      90      4       86      6       66      92
92      86      19      80      76      60      49
```

Initialized array “b” is now:

```
g       g
g       g
g       g
g       g
g       g
```

Enter values for b (10 of them):

```
a b c d e f g h i j
a       b
c       d
e       f
g       h
i       j
```

The element (2, 1) of array “a” is: 90

Changed element (2, 1) to -1 :

```
89      13      54      19      27      30      76
96      22      21      7       97      26      43
64      -1      4       86      6       66      92
92      86      19      80      76      60      49
```