

1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Lab06, put each programming exercise into its own subdirectory of this directory, zip the entire Lab06 directory up into the file Lab06.zip, and then submit this zip file to Lab #6.

Make sure that you:

- Follow the coding and documentation standards for the course as published in the MyClasses page for the class.
- Check the contents of the zip file before uploading it. Make sure all the files are included.
- Make sure that the file was submitted correctly to MyCLasses.

All non-templated class structures are to have their own guarded specification file (.h) and implementation file (.cpp) that has the same name as the class. All templated class structures are to be guarded and written entirely in their (.h) file. No inline coding in the class specification. In addition you must create a make file that compiles and links the project on a Linux computer with a Debian or Debian branch flavor.

2 Programming Exercises

These exercises are a bit of a continuation and combination of several previous labs, homeworks, and class discussions. Computer Science II is a course that begins your investigation of data structures, most of which are different types of containers for other data. So making these as efficient and general as possible is one of the primary goals for this course.

1. Start with the Array2D class from Lab #4 and Homework #3, some of the functionality from the Matrix class in Homework #4 may also come in handy. You will be templating this class so that it can store any data type. Hence creating a resizeable two-dimensional array that also stores its size and hence would be more convenient to use than your standard c++ two-dimensional array. as well as being stored in the heap.

- The data is be in the private section, all functions are public.
- There will be 4 data members,
 - The pointer (of pointers) to store the data.
 - rows that will store the number of rows the array is storing.
 - cols will be the number of columns the array is storing.

- A default value of the templated type. This default value will be used to populate empty cells of the array and be used as a return value from the get function if the indexes are out of bounds.
- The constructor will take in 3 parameters, the r , c , and defval. None of these should be default parameters, so on construction the programmer must specify them all. r is the number of rows, c is the number of columns, and defval is the value that the initial array will be populated with. This default value is to also be stored as the default value member variable for use in the get function.
- The destructor will free the memory allocation made for the array.
- Create appropriate overloads of a copy constructor and assignment operator.
- display: It is to take in a single integer parameter and use it as the column width in the printout of the array to the console. So `A.display(4)` will give each column 4 spaces to display and the values will be right justified in the column. If the width is less than or equal to 0 the array should be printed with a single space between the entries for each row, e.g. a call like `A.display(-3)`. Also if the parameter is not specified, `A.display()`, array should be printed with a single space between the entries for each row.
- getRows: Will simply return the number of rows being stored in the array.
- getCols: Will simply return the number of columns being stored in the array.
- set: Will take in three parameters, the first is the row index, the second is the column index, and the third is the value to be stored in that position in the array. If the index is out of the allocated memory bounds the program should print the message `Index out of bounds, no change to the array.` to the console and not update the array in any way.
- get: Will take in two parameters, the first is the row index, and the second is the column index. The function will return the value in that cell of the array. If the index is out of the allocated memory bounds the program should print the message `Index out of bounds, returning default value.` and it should return the default value that was specified in the constructor and is being stored as a member variable.
- resize: Will take in two parameters, the first is the new number of rows, and the second is the new number of columns. This function resizes the array to be an $r \times c$ array. If either dimension is less than 1 reset it to 1. This function is to also take the contents of the array before this call and preserve the cell values when the array is resized. So if the array is 3×3 and you resize it to 3×2 then the values in the first two columns are to be preserved. If one or both dimensions are larger than currently in the array the default value should be populated into the new cells.
- Overload the `[]` operator to return the row at the specified index. If the row index is outside of the allocated range the function should return the `nullptr`.
- Overload the `==` operator that will return true if the arrays are the same size and contain the same elements in each of the corresponding cells and false otherwise.

- Overload the `!=` operator that will return true if the arrays are either of different sizes or if any of the corresponding cells are not equal. It should return false if the two arrays are equal.
- Overload the stream out operator `<<` to print the array on one line. The rows are to have brackets around them and the entire array is to have brackets around it. The entries are to be separated by a single space. For example, if the array `M` is

```
4 4 7 2 4
2 4 0 6 8
7 2 9 8 8
```

then `cout << M;` would print `[[4 4 7 2 4] [2 4 0 6 8] [7 2 9 8 8]]`. One point of caution here that is not covered in the text is using friend functions with templates. We will not discuss this in general but give you some syntax for this particular operator. If in your specification for the class you use the following for the stream out operator (assuming that your template class variable is called `T`),

```
friend ostream &operator<<(ostream &, const Array2D<T> &);
```

you will get the following compile error.

```
warning: friend declaration `std::ostream& operator<<(std::ostream&,
const Array2D<T>&)' declares a non-template function
[-Wnon-template-friend]
```

This is because the `T` in the friend function and the `T` in the class are “interfering” with each other. There are a number of ways around this (with pros and cons on each approach) but here is the one we will use. In the specification for the class use the following for the stream out. Yes, another template inside the one for the class. Note that template name is `U` and not `T`.

```
template <class U>
friend ostream &operator<<(ostream &, const Array2D<U> &);
```

Then in the implementation you would go back to using the `T` template name, specifically,

```
template <class T>
ostream &operator<<(ostream &strm, const Array2D<T> &obj) { ... }
```

For example, the following main program will give the output below.

```
#include <cstdlib>
#include <ctime>
#include <iostream>

#include "Array2D.h"

using namespace std;

void div();
```

```
int main() {
    srand(time(0));

    Array2D<int> A(3, 5, -1);
    Array2D<string> B(3, 3, "George");

    A.display();
    cout << endl;
    B.display();

    div();

    for (int i = 0; i < A.getRows(); i++)
        for (int j = 0; j < A.getCols(); j++)
            A.set(i, j, rand() % 100);

    Array2D<int> C = A;

    A.display(5);
    cout << endl;
    C.display(5);
    cout << endl;

    A.set(2, 1, 123);
    A.set(1, 3, -15);
    A.display(5);
    cout << endl;
    C.display(5);

    div();

    A.set(20, 1, 123);
    cout << A.get(2, 2) << endl;
    cout << A.get(2, 20) << endl;
    cout << A << endl;

    div();

    A.resize(5, 6);
    A.display(5);
    cout << endl;
    A.resize(3, 6);
    A.display(5);
    cout << endl;
    A.resize(4, 5);
    A.display(5);

    div();

    for (int i = 0; i < A.getCols(); i++)
        A[A.getRows() - 1][i] = i + 1;

    A.display(5);

    div();

    C = A;

    A.display(5);
    cout << endl;
    C.display(5);

    div();
```

```

if (C == A)
    cout << "Arrays are equal." << endl;
else
    cout << "Arrays are not equal." << endl;

A[0][0]++;

if (C == A)
    cout << "Arrays are equal." << endl;
else
    cout << "Arrays are not equal." << endl;

if (C != A)
    cout << "Arrays are not equal." << endl;
else
    cout << "Arrays are equal." << endl;

return 0;
}

void div() { cout << "\n-----\n\n"; }

```

Output:

-1 -1 -1 -1 -1
-1 -1 -1 -1 -1
-1 -1 -1 -1 -1

George George George
George George George
George George George

89	13	60	76	95
93	44	65	66	37
20	77	42	52	72

89	13	60	76	95
93	44	65	66	37
20	77	42	52	72

89	13	60	76	95
93	44	65	-15	37
20	123	42	52	72

89	13	60	76	95
93	44	65	66	37
20	77	42	52	72

Index out of bounds, no change to the array.
42
Index out of bounds, returning default value.
-1
[[89 13 60 76 95][93 44 65 -15 37][20 123 42 52 72]]

89	13	60	76	95	-1
93	44	65	-15	37	-1
20	123	42	52	72	-1

```
-1 -1 -1 -1 -1 -1  
-1 -1 -1 -1 -1 -1
```

```
89 13 60 76 95 -1  
93 44 65 -15 37 -1  
20 123 42 52 72 -1
```

```
89 13 60 76 95  
93 44 65 -15 37  
20 123 42 52 72  
-1 -1 -1 -1 -1
```

```
-----  
89 13 60 76 95  
93 44 65 -15 37  
20 123 42 52 72  
1 2 3 4 5
```

```
-----  
89 13 60 76 95  
93 44 65 -15 37  
20 123 42 52 72  
1 2 3 4 5
```

```
-----  
89 13 60 76 95  
93 44 65 -15 37  
20 123 42 52 72  
1 2 3 4 5
```

Arrays are equal.
Arrays are not equal.
Arrays are not equal.