Computer Science II – Data Structures and Abstraction (CS23001)

Lab 4: More on Pointers and Dynamic Arrays

Objectives:

In this lab you will continue practicing the use of pointers and of dynamic arrays. The goal of this lab is to learn the following:

- Learn about pointers and dynamic variables
- Learn about dynamic arrays
- Learn about destructors, and copy constructors

Lab Submissions:

Please submit one Zip file containing all Source Code organized in separate folders along with evidence of working code, online through Blackboard by the end of the day on next Friday. No email submissions will be accepted. Name the Zip folder *yourname_Labx* (i.e. *AGuercio_Lab2*) Please ensure that your source code is properly organized. Create separate folders for each exercise and ensure that all needed files (headers, source, makefile) are included in each folder, Use exercise numbers as folder names. Run each exercise to test your code and save a screen shot showing the working code. Zip all folders together with screen shots before submitting it online.

Your code must have proper indentation, appropriate comments, pre and post conditions, and test cases. Test cases that you have used to test your program should be included as a comment on top of your driver file. Be sure you choose the **Test Cases** according to the "Testing" rules covered both in class/slides and in your book. Also be sure that you always separate the class declaration from its implementation and you create a Makefile for its compilation. This requirement MUST be satisfied in **every lab** of this course.

Lab Deliverables:

- Complete working source code of each exercise (where it applies).
- Screenshot(s) of each exercise to show that the code works.

Lab Grading:

The grade for each lab is based on lab attendance and working source code. Lab attendance and submission of Source Code along with Screen shots are mandatory for each lab. Grade distribution includes: Source code (75%), use of proper indentation and commenting of the code (5%), pre-conditions and post-conditions (5%), test cases (5%) and the evidence of working code in the form of screen shots (10%).

Good Luck!

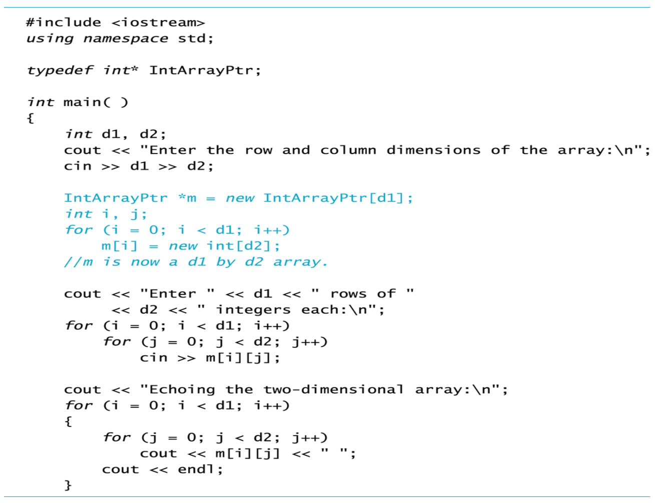
A dynamically allocated matrix

A matrix can be allocated dynamically by creating an array of pointers each pointing to an array of integers.

A 2-dimensional dynamic array is implemented as a dynamic array of pointers where each pointer points to a dynamically allocated array of a certain type. The size of the array of pointers corresponds to the number of rows of the matrix. The size of the dynamically allocated arrays pointed by the pointers corresponds to the number of columns of the matrix. The type of these arrays corresponds to the type of the elements of the matrix.

The following picture shows the code that implements a 2-dimensional dynamic array.

A Two-Dimensional Dynamic Array (*part 1 of 2*)



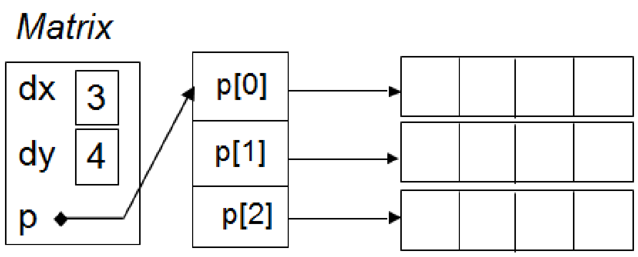
```
#include <iostream>
using namespace std;

typedef int* IntArrayPtr;

int main( )
{
     int d1, d2;
     cout << "Enter the row and column dimensions of the array:\n";
     cin >> d1 >> d2;

     IntArrayPtr *m = new IntArrayPtr[d1];
     int i, j;
     for (i = 0; i < d1; i++)
         m[i] = new int[d2];
     //m is now a d1 by d2 array.

     cout << "Enter " << d1 << " rows of "
          << d2 << " integers each:\n";
     for (i = 0; i < d1; i++)
         for (j = 0; j < d2; j++)
             cin >> m[i][j];

     cout << "Echoing the two-dimensional array:\n";
     for (i = 0; i < d1; i++)
     {
         for (j = 0; j < d2; j++)
             cout << m[i][j] << " ";
         cout << endl;
     }
```

Exercise 4.1

Write a driver function which initializes two rectangular matrices and multiplies them together. Remember that two matrices can be multiplied only if the number of columns of the first matrix is equal to the number of rows of the second matrix (i.e. A[n,m] X B[m,p] = C[n,p]).

A matrix class

Consider a matrix of **doubles**. In order to design a C++ class called Matrix that implements such matrix you need three private members which describe the two dimensions of the matrix and the pointer to an array of pointers.



Exercise 4.2

Create a Matrix class and implement the following member functions:

- The constructors and the destructor
- **getSize( )** which returns the size of the matrix;
- **setValue(int position, int value)** which sets the value in the matrix at given position;
- **getValue (int position)** which returns the current value at given position;
- an **add** method which adds two matrices together;
- a **subtract** method which subtract two matrices together;
- a **multiply** methods which multiplies two matrices together, if possible;
- an **overlap** function which overlaps two matrices together. The augment function takes two matrices A and B of size m1xn1 and m2xn2, respectively, and produce a new matrix C of size mLxnL, where mL (resp. nL) is the largest between m1 and m2 (resp. n1 and n2). The values of C will either come from the original matrices by overlapping matrix B over matrix A. Wherever a value is not available it initialized to default value 0.
  For example: If you overlap two matrices A of size 4x2 and B of size 3x5, the resulting matrix C is of size 4x5.

A=	2 3	B=	8 2 6 5 1	C = A overlap B =	8 2 6 5 1
	1 3		3 9 1 2 3		3 9 1 2 3
	6 3		5 7 3 1 4		5 7 3 1 4
	7 9				7 9 0 0 0

Here is another example. Consider matrix A of size 2x3 and matrix B of size 3x1, the overlapped matrix C is of size 2x3.

A =	2 3 5	B =	8	C = A overlap B =	8 3 5
	1 3 4		3		3 3 4
			5		5 0 0

While the numerical values of the example are integers, the class definition should be general enough to consider different possible types.

Test your class by writing a driver function that performs the following actions on the matrices A, B, and C given in the example above:

1. initialize two matrices A and B
2. add A and B together and then subtracts them
3. print both input matrices and the resulting matrix after addition and subtraction
4. print both input matrices and the resulting matrix after multiplication, if possible. If not possible explain the reason of the error
5. overlap the two matrices by using the **overlap** function.
6. display both input matrices and the output matrix produced by the overlap function.

Congratulations!

You have just completed another Lab!