Homework 5

Homework 5 will focus on operator overloading. Some of the code is provided, and some of this code shows a couple of neat tricks we can do in C++, one involving a clever use of references and the other involving delegating constructors.

In this homework you will be developing a 2D Array class (Array2D), and you need to implement some of the functionality associated with such a class. In particular, you will implement the following functions.

Functions to implement:

Array2D(Array2D& other): A copy constructor. The storage of the initialized 2D array representing the actual array data should be different than the storage of the data of the other 2D array's array data. That is, you should do a deep copy. If you look at what is being done for the other constructors (see their description under "Provide functions and constructors") you'll save yourself some work in creating the storage for the array in the object being initialized.

~Array2D: a destructor for a 2D array. It needs to free up the data from the actual array data, contained in ary. Hint: see and understand *deleteAry(int** ary)* that is provided to you.

Array2D& operator=: assigns a 2D array to another 2D array. In the assignment a1 = a2; the storage of a1 data representing the actual array should be different than the storage of the data of the a2 array. That is, you should do a deep copy.

Remember that C++ (like C) supports chained assignment, that is, you can have an assignment of the form a1 = a2 = a3. The semantics of this are that first a2 = a3 is executed. a3 is assigned to a2, and what is assigned to a2 is returned from the "=" operator. This value is then passed into the assignment to a1, and a1 has the same value as a2. This is done by having operator= both update the value of the *this* operand, and to return a reference to the *this* object.

bool operator==: compares two 2D arrays for equality. Two 2D arrays are equal if they have the same number of rows and columns and all of their elements are equal. Otherwise they are not equal.

Array2D& operator*: perform a 2D matrix multiply. Check for the arrays have dimensions that are compatible to being multiplied. Return an array filled with -1s that is the size of the *this* array if this is the case.

std::ostream operator<<: put the elements of the array represented by a Array2D on the output stream. The file output.txt shows how the array output should work.

int getNumRows( ), int getNumCols( ): getters that get the number of rows and columns of the array being represented.

Provided functions and constructors.

I have provided some functions and constructors for you, either because they are illustrative of useful tricks and techniques, or because they are not really central to operator overloading and there's no reason for 100 of you to implement these over and over again. I have also provided the HW5.cpp file that contains the main function, which should not need changing.

Array2D::Array2D(int numR, int numC) : *Array2D(numR, numC, 0, false)*: This constructor is used to create a 2DArray where the elements are initialized using default values, which in this case is to initialize each element to the value of its row index. It delegates to another constructor, called in the initializer list (shown in italics) to do the actual initialization of the array elements. Note that the constructor that is delegated to must be called in the initializer list.

Array2D::Array2D(int numR, int numC, int val) : *Array2D(numR, numC, val, true)*: This constructor takes a default value (val) to initialize the array elements. Like the constructor above, it delegates to another constructor, called in the initializer list (shown in italics) to do the actual initialization of the array elements.

Array2D(Array2D::Array2D(int numR, int numC, int val, bool useVal): a private constructor called by the previous two constructors that actually initializes the array. By delegating to this constructor we don't have to duplicate the code of setting up the storage for the 2D array.

int Array2D::operator( )(int row, int col) const and **int& Array2D::operator( )(int row, int col)**. C++ distinguishes between these based on whether the *this* pointer needs to be passed to a const parameter, or not. If *this* doesn't need to be passed to a const parameter, the version that returns a reference to an int (int&) is called, otherwise the one that returns an int is returned.

These functions are necessary because while we can overload the "[ ]" operator, there is no "[ ][ ]" operator in C++. "[ ][ ]" basically is the application of "[ ]" to an array of pointers, followed by application of another "[ ]" to what is returned by the first "[ ]". Thus we use the "( )" operation to do 2D indexing, allowing us to write things like *array(i,j)* where *array* is a 2DArray object or reference.

Interestingly, we can use *the int& Array2D::operator( )(int row, int col*) to both read and assign to an element of the array data. This is because of the int& that is returned. If you understand why this works, you probably have a pretty good grasp of how references work. If you don't, I'll provide an explanation of how this works after the homework is due, but am not going to do it now so as to give you a chance to figure it out on your own. In any case, I've written it and it is only used in the HW5.cpp file, so not understanding it immediately shouldn't make this homework harder to do.

bool sameSize: a helper array used to determine if two arrays are equal.

Void Array2D(int ary):** These free up the storage so that you can see how to call deletes on arrays of pointers. It is private because it only works on the private ary variable whose layout should not be visible outside of 2DArray, so not inheriting class should really know how to override this. I've made it non-virtual so that even if a class that inherits from Array2D tries to override it, we won't call that overriding function from code in the Array2D class.

What should be turned in?

The *userid* directory should be renamed with your userid, i.e., your Purdue login name. This directory should contain main.cpp, Array2D.cpp, Array2D.h and a Makefile such that executing "make" in the *userid* directory will make and run your program. The directory should not contain .o files or executable files. Zip up this file (use .zip files, *not .rar or .7z file*s, if at all possible, and if not at all possible let me know why) and turn it in to Brightspace.