**Homework Exercise: Wrapping a Dynamic Array**

**Downloads**

Download the starter files for this project.

The file will have both Code::Blocks and Visual Studio projects within it, and all the source code you need.

**Rules**

**Open book / Open note:**

Do research for this assignment as-needed. Use your textbook, the web, class example code, etc.

**Solo work**

This is a solo assignment and you should be working on this alone. This means...:

✓ You can verbally brainstorm with others to discuss the problem and techniques for problem solving.

✓ Your work must be your own work.

✓ Email the instructor, or ask in the Slack channel, if you are stuck or need clarification on the instructions.

✗ You cannot give your code to somebody else for them to "verify their work" with (more often than not they just copy it, and then you both get 0%!)

✗ You cannot copy another student's work. I will use a diff tool to compare student assignments (and code online). If there is obvious copying, all people involved will receive a 0%.

* Team size: 1 person
  + You can brainstorm with others - that is, you can verbally discuss the problem.
  + You cannot copy another person's work. Code will be checked against other submissions with a diff tool. If a copy is found, both people will receive a 0 on the assignment.
* Open book / open note - do research as-needed.

**Overview**

Implementing a Dynamic Array wrapper will be similar to implementing a wrapper to a Static Array, with some exceptions. Now, we will have to be concerned with memory management, and making sure our array is the right size.

We no longer have a maximum size cap - the max size is whatever we can store in RAM. Since we are unlikely to hit this RAM cap, we won't be checking for these errors right now.

**Turn in**

Once you're done, upload all source files (.cpp, .hpp, .h) to the class dropbox and up on your GitHub page.

**SmartDynamicArray Specification**

The following members have been added (vs. the static array):

* SmartDynamicArray()
* ~SmartDynamicArray()
* GetMaxSize()
* Free()
* Resize()
* Resize( int )
* m\_arraySize has been added.

The following already-existing members will need to be updated:

* Push( string )
* Insert( int, string )
* Extend( SmartDynamicArray )
* IsFull()
* operator=
* m\_data is now a string\*.

The following members can stay the same:

* bool Pop();
* bool Remove( int index );
* string Get( int index ) const;
* int Size() const;
* bool IsEmpty() const;
* string operator;
* bool operator==( const SmartStaticArray& other );

Changes are listed below. For implementations of the original static array versions, see the [Static Array Wrapper project](https://github.com/Rachels-Courses/CS250-Data-Structures/tree/master/Assignments/Exercises/Static%20Array%20Wrapper).

**Functions**

**SmartDynamicArray()**

This now needs to initialize the following:

* m\_itemCount to 0
* m\_arraySize to 0
* m\_data to nullptr

**~SmartDynamicArray()**

This needs to call the Free() function.

**bool Push( const string& newItem )**

* INPUT: A new item to add to the array
* OUTPUT: True on success, False on failure

Push can stay mostly the same, *except* that if the array is full (check with IsFull()), then Resize() should be called before the insertion. There should be no reason for this function to return *false* now.

**bool Insert( int index, const string& newItem )**

* INPUT: The index to insert the new item, the new item to add to the array
* OUTPUT: True on success, False on failure

Like Push( string ), this is mostly the same.

* If the array is full, call Resize()
* To check if the index is out of bounds, use m\_arraySize instead of MAX\_SIZE now.

**bool Extend( const SmartStaticArray& other )**

* INPUT: A second array
* OUTPUT: True on success, False on failure

Extend should call Resize(), passing in *m\_itemCount + other.Size()* as the new size, prior to copying the other array onto the current array.

**bool IsFull() const**

* INPUT: None
* OUTPUT: True if full, false if not full

You can check to see if the array is currently full by comparing whether or not *m\_itemCount* and *m\_arraySize* are equivalent.

**SmartStaticArray& operator=( const SmartStaticArray& other )**

* INPUT: A second array
* OUTPUT: A reference to the **this** array

Before copying over the contents of the other array, we need to make sure the sizes match first.

Call Free() on the local SmartDynamicArray, and then Resize( int ) to the other array's size.

Then, everything can be copied over as before.

**void Free()**

* INPUT: None
* OUTPUT: None

This should free memory and update the pointer per best practices...

1. Check if the pointer m\_data IS NOT *nullptr*. If so:
   1. Delete the dynamic array
   2. Set the m\_data pointer to *nullptr*

Otherwise, nothing else has to be done.

**void Resize()**

* INPUT: None
* OUTPUT: None

This should call the other Resize( int )function, passing in the *m\_arraySize + 10* as the argument.

**void Resize( int newSize )**

* INPUT: The new size for the array
* OUTPUT: None

These are the steps to "resizing" your dynamic array:

1. Create a *new* dynamic array with the larger size.
2. Copy over the contents from the old array to the new array.
3. Free the memory being pointed to by the old array (m\_data)
4. Update m\_data's pointer to the new array
5. Update the m\_arraySize value to the new size

**Grading Rubric**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Grading Rubric | | | |
|  | Name: |  | | |
|  | Assignment: | CS 250, Wrapping a dynamic array HWEX | | |
|  |  |  |  |  |
|  | Breakdown | | | |
|  | Item | Description | Total % | Your Score |
|  | Builds & Runs |  | 50.00% |  |
|  | Resize |  | 10.00% |  |
|  | Clean Code |  | 5.00% |  |
|  | No logic errors |  | 5.00% |  |
|  | Push update |  | 5.00% |  |
|  | Insert update |  | 5.00% |  |
|  | Extend update |  | 5.00% |  |
|  | Free |  | 3.00% |  |
|  | Constructor |  | 2.00% |  |
|  | Destructor |  | 2.00% |  |
|  | IsFull update |  | 2.00% |  |
|  | operator= update |  | 2.00% |  |
|  | m\_data update |  | 2.00% |  |
|  | GetMaxSize |  | 2.00% |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | Totals |  |  |  |
|  |  |  | 100.00% | 0.00% |