**Homework Exercise: Wrapping a Static Array**

**Downloads**

Download the starter files for this project.

* Visual Studio project
* Code::Blocks project

Or you can download the files directly on GitHub:

* [main.cpp](https://github.com/Rachels-Courses/CS250-Data-Structures/blob/master/Assignments/Exercises/Static%20Array%20Wrapper)
* [SmartStaticArray.hpp](https://github.com/Rachels-Courses/CS250-Data-Structures/blob/master/Assignments/Exercises/Static%20Array%20Wrapper)
* [SmartStaticArray.cpp](https://github.com/Rachels-Courses/CS250-Data-Structures/blob/master/Assignments/Exercises/Static%20Array%20Wrapper)
* [Tester\_SmartStaticArray.hpp](https://github.com/Rachels-Courses/CS250-Data-Structures/blob/master/Assignments/Exercises/Static%20Array%20Wrapper)
* [Tester\_SmartStaticArray.cpp](https://github.com/Rachels-Courses/CS250-Data-Structures/blob/master/Assignments/Exercises/Static%20Array%20Wrapper)

**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**

The starter files contain the stub for the Smart Array, with functions that need to be implemented.

The project also contains tester files that you can use to judge whether your functions are working properly.

**Turn in**

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

**Explanation**

For this program, you will be building a smart static array.

Included in the files is a tester. This will test every function of your array implementation, and either pass or fail the tests. If a test fails, it gives a comment hint.

You will need to figure out the algorithms for the array functionality, but these testers will help you figure out if you're on the right path.

All you have to do is run the program, and the test is executed when main() starts.

**SmartStaticArray Specification**

The SmartStaticArray already has its member variables and functions declared.

The class contains a string array, **m\_data**, and an item count, **m\_itemCount**. There is also a global named constant, **MAX\_SIZE**.

The following functions are declared, but not implemented:

* SmartStaticArray();
* bool Push( const string& newItem );
* bool Insert( int index, const string& newItem );
* bool Extend( const SmartStaticArray& other );
* bool Pop();
* bool Remove( int index );
* string Get( int index ) const;
* int Size() const;
* bool IsFull() const;
* bool IsEmpty() const;
* string operator;
* SmartStaticArray& operator=( const SmartStaticArray& other );
* bool operator==( const SmartStaticArray& other );

Descriptions of the functions are below.

**Functions**

**SmartStaticArray()**

The constructor is automatically run when a SmartStaticArray is *instantiated*.

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

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

Push should insert an item to the *end* of the array. For example, the first item will go at 0, then the next one at 1, then 2, and so on.

It should return false if the array happens to be full. Otherwise, it will return true if the assignment was successful.

**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

Insert will allow an item to be inserted into the middle of the array, or at the end or at the beginning.

This should not overwrite existing elements; you will need to shift all existing elements over to make room for this new item.

**NON-CONTIUGOUS** items are not allowed - if you have an array of size 5, you cannot insert something at position 100. Make sure to return false in this case, without changing the array.

Make sure to validate the index: It shouldn't be negative, or MAX\_SIZE or greater. Return false if there is an invalid index given, without changing the array.

If the insert is successful, return true.

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

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

This function will take the second array, and append it to the end of **this** array.

If there isn't enough room to append the second array, then return false without modifying the **this** array.

After the extend, return true on success.

**bool Pop()**

* INPUT: None
* OUTPUT: True on succes, False on failure

This function will remove the item at the very end of the array.

Lazy-deletion: You don't actually have to remove the item; you just have to adjust *m\_itemCount*.

If the array is empty, return false and don't do anything. Otherwise, return true.

**bool Remove( int index )**

* INPUT: Index of item to remove
* OUTPUT: True on succes, False on failure

Remove the item at the given index. If there are any items to the *right* of this element, shift them over left to fill in the space so that the array remains continugous.

If the index is invalid (negative or MAX\_SIZE or greater), then return false without doing any modifications.

On success, return true.

**string Get( int index ) const**

* INPUT: Index of item to access
* OUTPUT: Value of the item accessed

Return the value of the item at the index given.

If the index is invalid, return an empty string instead. ""

**int Size() const**

* INPUT: None
* OUTPUT: Count of items in the array

Return the count of items stored in the array.

**bool IsFull() const**

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

**bool IsEmpty() const**

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

**string operator**

* INPUT: Index of item to access
* OUTPUT: Value of the item accessed

Basically the same functionality as Get(), but it is the subscript operator.

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

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

Set the **this** array equal to the second array's values.

Iterate through all the items of the second array, copying the values over to the **this** array.

Make sure to also update the *m\_itemCount*.

Return a reference to \*this afterward.

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

* INPUT: A second array
* OUTPUT: True if the second array is the same as **this**, or False if not.

Iterate through all elements of both arrays and compare the values.

If they are equivalent, then return true.

Otherwise, return false.

**Grading Rubric**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Grading Rubric | | | |
|  | Name: |  | | |
|  | Assignment: | CS 250, Wrapping a static array HWEX | | |
|  |  |  |  |  |
|  | Breakdown | | | |
|  | Item | Description | Total % | Your Score |
|  | Builds & Runs |  | 5.00% |  |
|  | Clean Code |  | 5.00% |  |
|  | No logic errors |  | 10.00% |  |
|  | Constructor |  | 2.00% |  |
|  | Push |  | 5.00% |  |
|  | Insert |  | 13.00% |  |
|  | Extend |  | 13.00% |  |
|  | Pop |  | 5.00% |  |
|  | Remove |  | 10.00% |  |
|  | Get |  | 5.00% |  |
|  | Size |  | 2.00% |  |
|  | IsFull |  | 2.00% |  |
|  | IsEmpty |  | 2.00% |  |
|  | Operator[] |  | 5.00% |  |
|  | Operator= |  | 8.00% |  |
|  | Operator== |  | 8.00% |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  | Totals |  |  |  |
|  |  |  | 100.00% | 0.00% |