CS20 – Section 093 – Fall 2019
Programming Assignment 6 – Due Sunday October 13 at 11:59PM

Stacks implemented via linked list of arrays (60 points)

In this assignment, you'll create a Stack ADT. It maintains strings as the data type (no template). Here is the Stack UI:

> **Stack name (number of elements);**
>
> Creates a stack of strings with name "name." Internally, each unit of memory allocation is "number of elements" elements. Users should be smart enough to choose a number of elements that matches their expected usage patterns.
>
> **void push(string)**
>
> Adds an element containing string to the top of the stack.
>
> **bool top(string &)**
>
> Retrieves the string in the top element. If the stack it empty, returns false.
>
> **bool pop(string &)**
>
> Retrieves and removes the top element. If the stack is empty, returns false.
>
> **destroy()**
>
> Releases any memory used by the stack.

<u>Implementation overview:</u>
Dynamically allocate an array of strings, with whatever size was specified by the user. Let's assume that the size was 10 elements. When the user pushes a value, add that value to the array. As the user pushes and pushes, the array fills up. When the array has 10 elements, it is full. The next time the user does a push, allocate a new array, and place the user's data into the first element of that new array. At this point, you have two arrays of 10 elements. The oldest array has all 10 of its elements in use, and the newest has only one element in use. Keep track of the arrays by using a linked list. It is a linked list of pointers to arrays. When you create a new array, you add a node to the list front of the list containing a pointer to that array. Only the most recent array (the one pointed to by the front node in the linked list) is "current." The front element of the linked list always points to the current array.

When the user pops an element, locate the current array, and copy the top element's data so that you can return the data to the user. Remove the element from the current array. If that causes the array to become empty, delete the memory allocated to the array, and remove the linked list entry for the array. Now the linked list contains one less element. The front of the list points to the current array.

The user can delete the memory used by the stack by using the destroy function at any time.

You'll write code to implement the Stack class and member functions described above. You may use either your linked list implementation from Assignment 2, or the STL list.

There are a few differences between this ADT and the Stack described by Malik in chapter 7:

- Malik's stack does not dynamically expand and contract as needed.
- Malik provides an "initializeStack" function which perfoms initialization functions. We put that logic into our ADT, simplifying the experience for the user.
- Malik provides an "isFullStack" function which lets the user know in advance if a forthcoming push will fail. Since our stack expends as necessary, this function is not required.
- Malik provides an "isEmptyStack" function. In our implementation, if a top or pop fails because of an empty stack, we let the user know via a boolean return value.
- Malik's top function terminates if the list is empty; we return a bool = false to indicate an empty stack
- Malik's pop only removes the top entry; is does not retrieve the data.

<u>Implementation details:</u>

```
Stack class members
    array size (specified by user when instantiating)
    linked list definition
    topElement (element number of the current top element of the stack)
```

The element number is logically a "pointer" to the top element. It always points to the place in the current array where the next element would go. Again, assume that the array size is 10. If the element number is 5, then a new element goes into array [5]. When the list is empty, the element number is zero. When the element number is 10, the array is full.

```
constructor
    allocate an array of the specified size
    create a linked list entry containing a pointer to the array
    set topElement to 0, indicating that the top of the stack is element 0 of the current array

push
    if the current array is full (topElement = array size)
        create a new array
        add a pointer node for this array to the linked list
        move the data to element 0 of the new array
        set topElement to 1
    else
        add data to the array at position {topElement]
        topElement ++

pop
```

```
    if the current array is empty (topElement = 0)
        if there is only one array allocated
            return false
        else
            delete current array
            remove linked list entry for array
            set topElement = array size - 1
            retrieve data from current array [topElement]
    else
        topElement - -
        retrieve data from current array [topElement]

destroy
    delete array
    remove linked list node for array
    set topElement = array size
```

Use a header file for your Stack class, a .cpp for its member functions, and a second .cpp for the main routine.

Design decisions

When retrieving data from the current array, or when storing data into the current array, you need the address of the array. You can always get the address by referring to the front element of the linked list. An alternative, somewhat more efficient method would be to add a member to the class that always contains the address of the current array. This this pointer is updated whenever you add or remove an array from the linked list.

This implementation always has at least one allocated array, even if it's empty. The initial array is allocated by the constructor, and only deleted when the user uses destroy. The pop logic takes this into account. When pop sees that an array has been emptied, it only deletes it if it isn't the only array. This way, the user can empty a stack completely, and then immediately repopulate it without any intervening functions. An alternative would be to change the logic so that push allocates an array whenever one is needed, and that pop removes an array anytime one is emptied. The stack could still be "active" even though it doesn't have any allocated memory.