CSE 232 Spring 2017

Programming Project #10

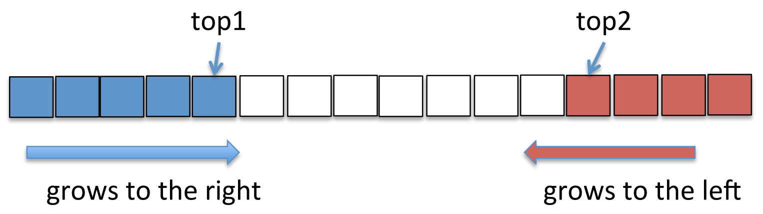
Assignment Overview

You will create a `BiStack` data structure, a data structure that maintains two stacks in a single array of dynamically allocated memory. The `BiStack` will be a ***template class*** that uses ***dynamic memory*** to store its contents.

This assignment is worth 60 points (6.0% of the course grade) and must be completed and turned in before 11:59PM on Monday, April 17th .

Background

A `BiStack` data structure is, as mentioned above, a single data structure that maintains two stacks simultaneously in a single, dynamically allocated, array. If you need two stacks, and there are other data structures that can implemented using two stacks, then a `BiStack` is a memory efficient way to represent them. It looks like the picture below:



The two stacks grow towards each other, and only when there is no memory left in-between are both stacks full.

A Different Specification

As previously, your job is to design the class to meet the specifications as provided by the main and test cases. However, our ***`BiStack` is a templated class*** so we must put all the information, the class declaration and the class definition, in the same file. Thus you will provide only `class-10.h` containing both as your assignment. You must create the .h file to meet the specs of the various test programs. The interface is described below.

Specifications

- the underlying data structure will be a single, dynamically allocated array of template type. This array can grow when the `BiStack` is full but only up to the provided maximum size indicated in the constructor.

- **constructors**
  - two args: two `size_t` which indicate, in order
    - initial array capacity. **default value** of 4
    - maximum array size. **default value** of 16

    - two arg: an `initializer list` and a `size_t` (in that order)
        - `initializer_list` of the template type fills the top1 side only **(not the top2 side)** of the BiStack
            - top2 side remains empty
        - the capacity, the size of the underlying array, is set to the size of the initializer_list
        - `size_t` is the maximum size: **default value** of 16
    - copy constructor
- **destructor**
- **`assignment operator.`** Used to assign one `BiStack` to another `BiStack`
    - make sure it does the appropriate copy and doesn't leak.
    - you decide on appropriate args and return vals.
- **`grow_and_copy`** .
    - ***private*** method, no args and no return
    - When the `BiStack` array is full, this method is called to grow the underlying array.
    - creates an array of double the present size, and ***fills it appropriately***
    - if the new size is less or equal to the max size
        - makes a new array of double size
        - copies the content of the original array to the new array ***in the correct location***
        - the two tops are updated
    - if the new size is larger than the max, throws an `overflow_error` and the underlying data structure is ***unchanged***
- **`operator<<`** prints the contents of the `BiStack`
    - prints the contents of the underlying array
    - prints the values of the two tops
    - as a friend, it should be ***defined inline*** as a part of the class declaration.
    - see tests for format. ***FOLLOW IT!!***
- **capacity** method
    - no args,
    - returns a `size_t` type, the present capacity (the present size) of the underlying array
- **size** method
    - no args
    - returns a `size_t` type, the number of elements stored in the underlying array
- **max** method
    - no args
    - returns a `size_t` type, the maximum size that the underlying array is allowed to grow to (value provided in constructors)
- **`push1`** and **`push2`** methods.
    - 1 arg of the template type, no return
    - pushes a new value onto the respective stack
        - stack size is one larger
    - if full, calls `grow_and_copy` to make more room.

        - if new size exceeds max, `grow_and_copy` throws an `overflow_error`
- **`top1`** and **`top2`** methods
    - no args, return of template type
    - if not empty
        - returns the top of the respective stack
        - otherwise throws a `underflow_error`
- **`pop1`** and **`pop2`** methods
    - no args, no returns
    - if not empty
        - removes the top element from the respective stack
            - the stack size is one smaller
        - otherwise throws a `underflow_error`
- **`empty1`** and **`empty2`** methods
    - no args, Boolean return
    - returns true if the respective stack is empty
        - false otherwise

Errors

Certain errors require that you throw an exception.

- `grow_and_copy` beyond max, throw `overflow_error`
- `pop` and `top` when empty, throw `underflow_error`

Anything else you need!!!

You are defining your class, so any other functions/members you might need, feel free to add them.

Important things to Note

Deliverables

`class-10.h` your source code solution containing declarations and definitions in one file, since this is a template. remember to include your section, the date, project number and comments.

1. Please be sure to use the specified file names
2. Save a copy of your file in your CSE account disk space (H drive on CSE computers).
3. You will electronically submit a copy of the file using the "handin" program: http://www.cse.msu.edu/handin/webclient

Assignment Notes

- Lab 12 should be a very big help (assuming you did it)
- the most difficult part is likely the `grow_and_copy` method. It is easy to copy the front of the old array to the front of the new (double sized) array. It takes some calculation to copy the back of the old array (the top2 side) to the back of the new (double sized) array.

- the test case inputs are just the case number. You can just type the test case number if you want to run a case.
- So that printing wasn't too weird, I initialized the array when it is allocated to the default value of the data type with empty brackets
  `T *ary_ = new T[size]{}`
  that initializes every value to the default. You could also use the fill method, using a `T()` as the default type (look it up)
- Feel free to use code provided in the class (such as my examples or the worksheets), but you should put a comment in the code to indicate where you got it.