

Due: Thursday, July 7  
Emphasis: Stacks

In this assignment we will extend the dynamic array implementation of the Stack class given in Figure 7.10 p. 344 and use the Stack to solve some problems. I have provided the **DStack** header and implementation file, as well as test driver, as a starting point for your modifications for this assignment.

Part 1, extension of **DStack** functionality: In part 1 we will be adding some member functions to the stack class.

- a) Write documentation, a prototype, and a definition for a member function *bottom()* for the DStack class that returns the bottom element of the stack. Add calls in test driver to test your *bottom()* function.
- b) Write documentation, a prototype and a definition for a member function *nthElement()* to retrieve the *n*th stack element (counting from the top), leaving the stack without its top *n* elements. Add calls in the test driver to test your *nthElement()* function.
- c) The current implementation of *push()* simply displays an error message and exits if we attempt to push an item onto an already full stack. Change the implementation of *push()* to instead cause the stack to grow whenever it is full. Your implementation should create a new dynamic array that is double the size of the current stack array whenever a push is attempted on a full stack. It will need to copy the elements from the old array to the new array, then free up the memory of the old array by calling *delete* on it. Add calls in your test driver again to test that your behavior is correct and that the stack grows appropriately whenever a *push* is attempted on a full stack.

Part 2, using **DStack** to solve a problem. In part 2 we will use the **DStack** data type in order to solve a simple problem.

- a) Use your **DStack** class from part 1 to create a function that reads a string, one character at a time, and determines whether the string contains balanced parentheses, that is, for each left parenthesis (if there are any) there is exactly one matching right parenthesis later in the string.

### Requirements

- 1) You must start with and modify the DStack class provided for you to implement the additions and changes in Part 1.
- 2) For Part 1, turn in your modified DStack.h, DStack.cpp and the DStackTestDriver.cpp that you changed to satisfy and test your changes for Part 1 a,b and c.
- 3) For Part 2, create a new .cpp file to implement the parenthesis matching problem. You should implement the parenthesis matching as a function, that accepts a string as a parameter. The function will need to create a single DStack object and use it to test the string to see if the parenthesis are matching. Your function should return a bool value that is true if the parenthesis matched up in the provided string, and that is false otherwise.
- 4) Test your function for Part 2 by making up a couple of string examples and calling your function to test them. Make sure you test some examples of strings with correctly matching parenthesis and mismatched parenthesis. Also test a string that has no parenthesis. A string with no parenthesis should return true from your function as being matched.