CSC2040 Data Structures and Algorithms, and Programming Languages

Practical 6

Wednesday 22 November 2017

Stacks

The learning outcome of this practical is to be confident in using a standard data structure (a stack class), and to be able to add new facilities to the library class.

The supplied folder Practical 6 Files contains the files:

ArrayStack.h
LinkedStack.h
StackNode.h

Part 1: Using the Stack classes for applications

For this part, you should first of all use the **ArrayStack** class for all your stacks.

Create your own project Practical6 with a test.cpp file. Copy the above three header files into the folder Practical6/Practical6, add them as Existing Items into your project, and #include them in the test.cpp file.

Note that you must #include <iostream> before #include the above three header files [why? There is a method to get around this, how?]

In the test.cpp file, create a main() function, which is used to perform the tests below.

(1) Above the main() function, write a template function:

void **setUpStack**(ArrayStack<T>* s, int num)

which takes an existing (empty) stack of type T, and pushes a sequence of num type-T values inputted from cin on to the supplied stack. In the main() function, call this function with a type-string stack, and push the seven days of the week (from Sunday to Saturday).
[Note: you have to pass the address of the stack to call this function]

(2) Above the main() function, write a template function

void **printStack**(ArrayStack<T>& s)

which prints the contents of the stack s, from the top to the bottom, but LEAVING THE STACK AS IT WAS. Call this function to print out the stack you constructed in (1) above. Then call it again to prove that you have not altered the stack!
[Note the differences between the two functions in (1) and (2), in the method of defining the parameter s, and in the method of passing s in making the Call]

(3) Write a template function above your main() function file called

void **removeAt** (ArrayStack<T>& s, int p)

which takes a type-T stack s and a position p, and removes the p^{th} item from the bottom in the stack. Remember: you should not edit the supplied stack classes! And don't be tempted to use an array. If there is no item at position p, then output an error message to the console. Test the effect of your function using printStack.

Now, replace the ArrayStack objects with the **LinkedStack** objects instead for your functions in (1) to (3) above. Does everything work the same?

Part 2: Adding functions to the Stack classes and their implementation

In this part you change from being an application developer to being a library developer.

Add a new public function to the ArrayStack and ListStack classes:

int **searchStack** (T item)

This function should return the position of the first occurrence of 'item' in the stack, starting from the top of the stack. It should not change the stack in any way. The bottom position is 0. If the item doesn't appear in the stack, you should return -1.

Write code in main() to test if your revised stack classes work properly.

Part 3: Challenge

This challenge is intended to stretch you a bit in your C++ programming.

Write a function which inputs a **postfix** expression from cin, represented as a sequence of character strings ending with a '**#**'. It should calculate the result as it goes along, and print the result.

void **evaluateExpression** ()

The sequence of character strings from cin defines the expression, with each character string being either an integer number, or an operator which can be '+', '-', '*' or '/'. Your function should read the sequence of character strings one string at a time, determine for each string if it represents a number or an operator, and perform each operation using a stack. It should print out the final answer (which should be the only remaining item on the stack) when you input a '**#**'.

Use the following expression (equal to 1360) to test your program:

(12 + 4) * ((8 - 3) * (2 * 11 - 15/ 3))

The postfix notation would be represented by a sequence of character strings as follows, to be typed in one string at a time from cin:

12 4 + 8 3 - 2 11 * 15 3 / - * * #