# Assignment 06 - Stack Template

[Re-submit Assignment](https://dvc.instructure.com/courses/27283/assignments/220080?module_item_id=434851)

This assignment involves the writing of 2 CPP's. To it's clear which CPP goes with which "part" of the assignment, make sure that you have some reference to the part number in your file names (like part1.cpp or testing.1.cpp for example).

### Part 1, Developing And Testing A Stack Template

Write a template, **MyStack.h**, to implement a LIFO stack. Here is the specification for its public interface:

class MyStack  
{  
 ...  
 MyStack( ); // may have a defaulted parameter  
 void push(const V&);  
 V& peek( );  
 void pop( );  
 int size( ) const;  
 bool empty( ) const;  
 void clear( );  
};

If you use dynamic memory (and you surely will!) be sure to include the three memory management functions as public members, too.

You may implement your MyStack as arrayed or as linked -- your choice.

Fully test your template in a test driver CPP named as you wish, remembering to include all the tests we've learned about in this class. Then use the H file in the following application:

### Part 2, An RPN Calculator App

Use your stack template from part 1 to apply "reverse Polish notation" ([RPN (Links to an external site.)Links to an external site.](https://dvc.instructure.com/courses/27283/pages/Reverse_Polish_notation" \o "" \t "_blank)) as it was used in the earliest electronic calculators. In the 1960's, digital processors and memory were expensive, big, and slow. Algorithms for parsing, that we have in modern calculators, were just not feasible because of the memory and processing power required.

Calculators like the [HP-35 (Links to an external site.)Links to an external site.](http://www.hpmuseum.org/hp35.htm" \o "" \t "_blank) simplified math operations by applying RPN. They worked like this -- [operands (Links to an external site.)Links to an external site.](https://dvc.instructure.com/courses/27283/pages/Operand" \o "" \t "_blank) for plus, minus, multiply, and divide got entered and stored in a stack. Then [operations (Links to an external site.)Links to an external site.](https://dvc.instructure.com/courses/27283/pages/Operation_(mathematics)" \o "" \t "_blank) got entered, that would pop the top two values from the stack, do the operation, and push the result back to the stack. So one plus one worked like this: 1 [ENTER] 1 [ENTER] -- now there are two values in the stack. Then + [ENTER] -- the two 1's are popped, added, and 2 gets put onto the stack. Final stack size, 1, and no parsing required!

Stack size was large enough for more complex calculations, like (1 + 2) / (4 + 5). Since the result of an operation gets put onto the stack and not just simply displayed, here's how this more complicated expression worked: 1 [ENTER] 2 [ENTER] + [ENTER] -- now 3 is on the stack, where it will remain while this is done: 4 [ENTER] 5 [ENTER] + [ENTER], which puts 9 onto the stack above the previous 3. Stack size, 2. Now just one more operation: / [ENTER], and 0.3333333 replaces the popped 3 and 9 on the stack. To recap, 1 [ENTER] 2 [ENTER] + [ENTER] 4 [ENTER] 5 [ENTER] + [ENTER] / [ENTER].

Write your appas a C++ console program. In a loop, process input from the user. If a plus (**+**), minus (**-**), multiply (**\***), or divide (**/**) symbol is entered, perform the operation with the two values at the top of the stack, and push the result onto the stack. If a Q or q is entered, break from the loop. Otherwise, apply **atof** to the entry (without checking to see if it's a number) and push its result onto the stack. Allow for floating point values.

DO NOT worry about division by zero or any other numeric validation. DO avoid popping from the stack when the stack is empty -- for example, if the user enters a plus operator, and there are fewer than two values in the stack, simply ignore the user's request and loop back to wait for the next input.

You may allow the user to type multiple entries on the same line of input, space-separated (before pressing ENTER), if you wish. Like 1 2 + 4 5 + / [ENTER]

Include the current stack contents in the input prompt (see below). Formatting of output precision is not required or recommended.

**Example (computer prompts in bold).** Your program should match these RESULTS, formatted as you choose.

|  |  |  |
| --- | --- | --- |
| *Example: five plus six* **Enter:** 5 [ENTER] **Enter: 5** 6 [ENTER] **Enter: 6 5** + [ENTER] **Enter: 11** *Example: ten minus one* **Enter:** 10 [ENTER] **Enter: 10** 1 [ENTER] **Enter: 1 10** - [ENTER] **Enter: 9** Q [ENTER] *Example: one divided by two* **Enter:** 1 [ENTER] **Enter: 1** 2 [ENTER] **Enter: 2 1** / [ENTER] **Enter: 0.5**  Example: add with empty stack **Enter:** + [ENTER] **Enter:** |  | *Example: Pi times the radius squared* **Enter:** 3.14159 [ENTER] **Enter: 3.14159** 18 [ENTER] **Enter: 18 3.14159** 18 [ENTER] **Enter: 18 18 3.14159** \* [ENTER] **Enter: 324 3.14159** \* [ENTER] **Enter: 1017.88** *Example: (1+2)/(3+4)* **Enter:** 1 [ENTER] **Enter: 1** 2 [ENTER] **Enter: 2 1** + [ENTER] **Enter: 3** 3 [ENTER] **Enter: 3 3** 4 [ENTER] **Enter: 4 3 3** + [ENTER] **Enter: 7 3** / [ENTER] **Enter: 0.428571**  Example: add with one value in stack **Enter:** 1 [ENTER] **Enter: 1** + [ENTER] **Enter: 1** |

**Order matters.** Note that 10 [ENTER] 1 [ENTER] - [ENTER] is 9, not -9. and 1 [ENTER] 2 [ENTER] / [ENTER] is 0.5, not 2. Make sure your program gives the same numeric results as in the samples above.

### Submit 4 Files

Submit your test driver CPP, your app CPP, and your template's H file for grading.

Also, Submit a word document with your code copied and pasted PLUS one or more screen shots of the output of your app.