[LW] Stack Calculator

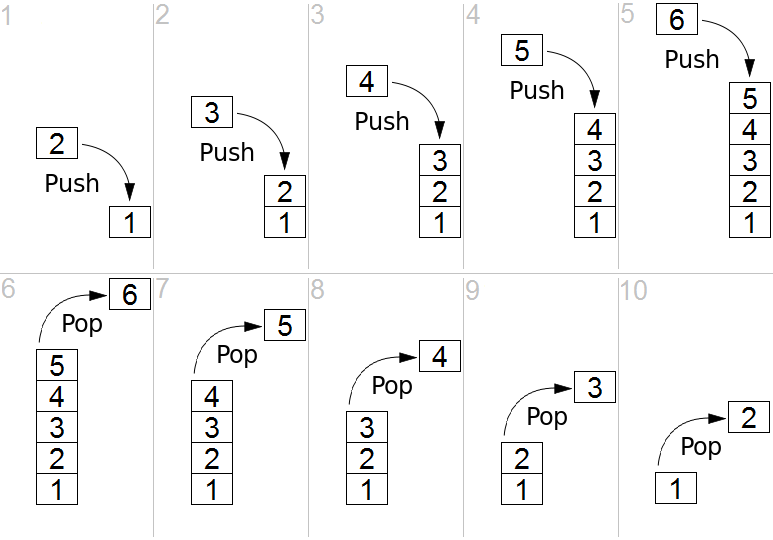
For this labwork, you will be creating a calculator that uses a specific mathematical notation called Reverse Polish Notation (RPN) and that uses a stack data structure.

* Refer to **Background** for a brief overview of the stack data structure that you will be using in your code..
* Refer to **Grading** for details of satisfying labwork completion. Work with others in your group on the labwork, but submit individually on Mimir.
* Refer to **Instructions** for information on completing the different tasks in the labwork.

# Objectives

* Correctly code a stack implementation using step-by-step pseudocode.
* Correctly code a stack-based mathematical algorithm using step-by-step pseudocode.
* Correctly apply memory management steps for an algorithm that uses the heap.
* Use I/O streams to properly read user input and output a correct arithmetic result.

# Background: Stack



Simple representation of a stack runtime with push and pop operations. (<https://en.wikipedia.org/wiki/Stack_(abstract_data_type)#/media/File:Lifo_stack.png>)

A *stack* is an abstract data type that serves as a collection of elements and has two principal operations:

1. push : add an element at the top of the stack
2. pop : remove the element at the top of the stack

A stack may also have a peek operation, which gives access to the top element without removing it.

# Grading

As stated in the syllabus, you must complete both items below to receive credit for this week's lab. **The Teaching Assistants will strictly enforce attendance that is described in this section at the start and end of your lab.**

* complete the labwork
* attend your assigned lab

## Submission Grade

You must submit the completed labwork as two files on Mimir:

* Stack.cpp: the data structure code to be used in your stack calculator
* RPN.cpp: the code for the stack calculator user interface

## Attendance Grade

* Your TA will mark your attendance at the start of class, and will confirm your attendance after your group shows the minimum required completed work to be checked by your TA.
* If you do not attend your lab at the start of class or if you do not receive confirmation from your TA when your group submits, then your attendance will not be recorded.

## Makeup Work

Before you can do any make up work, you must provide your instructor with any documentation for your excused absence.

# 

# Instructions

Below are the full set of instructions for this labwork. **The Teaching Assistants will provide assistance in this section while your group is working on the labwork.**

## Task 1: Implement a Stack (of doubles)

Read Stack.h, which contains the definition of the Stack type and declarations of the Stack operations which you must implement.

Pushing and popping should be done at the back (last element) of the array:

Let A = [1, 2, 3, ], capacity = 4, length = 3

Push 4

Now A = [1, 2, 3, 4], capacity = 4, length = 4

Pop (returns 4)

Now A = [1, 2, 3, ], capacity = 4, length = 3

Push 5

Now A = [1, 2, 3, 5], capacity = 4, length = 4

Resizing the array should double the capacity:

Let A = [1, 2, 3, 4], capacity = 4, length = 4

Push 5

Now A = [1, 2, 3, 4, 5, , , ], capacity = 8, length = 5

## Task 2: Implement a 4-function RPN Calculator

**Reverse Polish Notation (RPN)** is a mathematical notation in which operators follow their operands (also called *postfix notation*).

* For example, the arithmetic expression “1 + 2” would be written as “ 1 2 +” in RPN.

One of the benefits of RPN is that parentheses are not required.

* For example, the arithmetic expression “1 + 2 \* 3 / 4 - 5” is ambiguous without some assumption about the order of operations.

According to the standard order of operations (PEMDAS), the expression means:

* (1 + ((2 \* 3) / 4)) - 5

In RPN, the equivalent expression is:

* 1 2 3 \* 4 / + 5 -

The RPN algorithm is simple and uses a stack to store intermediate values of the computation:

1. Read numbers and push onto the stack until you read an operator (one of +, -, \*, /) or =
   1. Hint: read a string (e.g. std::string or char[]), then check if the first character is a digit or ‘.’ (dot). If so, then the string is an operand (convert to double with stod(std::string) or atof(char[])), otherwise the string is an operator (use switch to handle the different cases).
      1. Or, attempt the conversion to double and catch the exception if the string is not a double (therefore it is an operator).
2. If an operator (+, -, \*, or /) is read,
   1. pop the top two elements of the stack into variables for right and left operands
      1. First pop goes into right
      2. Second pop goes into left
   2. Do the operation and push the result onto the stack
3. Repeat steps 1--3 until ‘=’ is read
4. Pop the result from the stack

You should read from standard input and print to standard output.

# Constraints

1. Your source code file must be named:
   1. Stack.cpp: implements Stack
   2. RPN.cpp: implements 4-func RPN calc
2. You may #include
   1. <cmath>: for using the **NAN** constant
   2. <iostream>:for std::cin, cout, endl
   3. <string>: for std::string, stod(), isdigit()
   4. “Stack.h”: for using your Stack.cpp functions
   5. and nothing else.
3. You will submit **only**:
   1. Stack.cpp: implements Stack
   2. RPN.cpp: implements 4-func RPN calc
4. The starter code contains:
   1. “Stack.h”
5. Your program should not leak memory
   1. Every new should have a corresponding delete (or delete[])

## 

## Examples for RPN Calculator

**Dark Red Bold** text is user input.

### Input

**1 2 3 \* 4 / + 5 - =**

### Expected Output

-2.5

### Input

**1 2 + 3 \* 4 / 5 - =**

### Expected Output

-2.75

### Input

**10 10 10 \* \* 59 + 1024 8 \* \* 19 - 10000 / =**

### Expected Output

867.531

### Input

**10 0 5 - - =**

### Expected Output

15

### Input

**1 2 3 4 5 6 7 8 9 10 + + + + + + + + =**

### Expected Output

54

### Input

**3 2 ^ =**

### Expected Output

[ERROR] invalid operator: ^