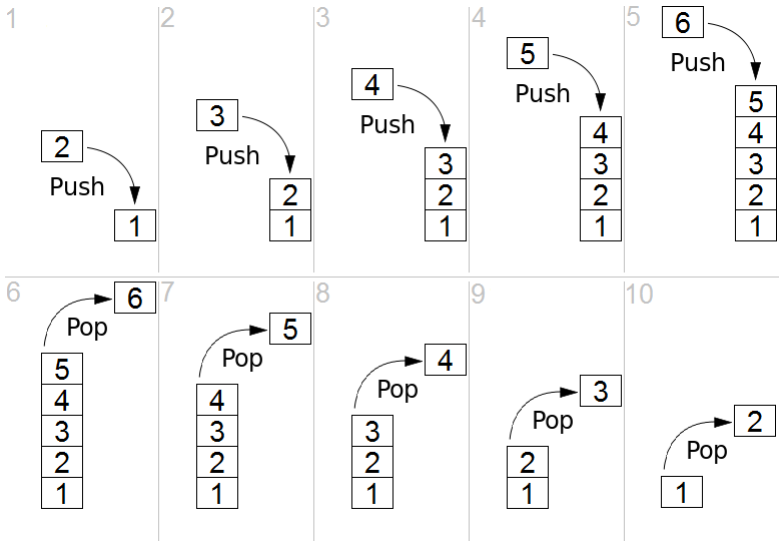
Programming Project #3
EGRE246 Spring 2018
Expression Stack Machine

1 Introduction to Stacks

A *stack* is an ADT container whose elements can only be accessed from one end (referred to as the 'top'). The primary operations on a stack are:

`void push(const value_type& entry);`	pushes `entry` onto the stack at the top
`value_type pop();`	pops a value off of the top of the stack and returns it; asserts first that the stack is not empty
`value_type top();`	returns the item at the top of the stack without removing it; asserts first that the stack is not empty
`bool isEmpty() const;`	returns true if the stack is empty, false otherwise



As you will need a stack class for this project, you are to add these 4 routines to the `Sequence` class presented in class and then rename it `SeqStack` (i.e. you will have a `SeqStack`*XXXX*`.cpp` file (where *XXXX* is the last digits of your student id) that implements `SeqStack.h` (which will be given to you on the class web pages). Note that all of the routines listed above must set the current item to the last item in the sequence before and after the operation.

2 Project Overview

A *stack machine* (SM) is a computer or model of computation where the machine's memory takes the form a stack. Our SM instructions can only operate on (integer) values stored on the stack. For example, executing the following SM code (7 instructions) would print the value 50:

`LDC 2` – load (push) constant 2 onto stack
`LDC 3` – load constant 3 onto stack
`ADD` – pop top 2 items, add them, push result onto stack
`LDC 10` – push constant 10 onto stack
`MUL` – pop top 2 items, multiple them, push result onto stack
`OUT` – pop top of stack and output to screen
`HLT` – halt the machine

2.1 Instruction Set

For this project you will implement a SM interpreter for the following instruction set:

CLR	clear the stack to empty
HLT	halt the stack machine
NEG	negate the top-of-stack
DUP	duplicates item on the top of stack
ADD	add next-to-top to top-of-stack, leave result as top-of-stack
SUB	subtract top-of-stack from next-to-top, leave result as top-of-stack
MUL	multiply next-to-top by top-of-stack, leave result as top-of-stack
DIV	divide next-to-top by top-of-stack, leave result as top-of-stack;
LDC n	push integer n onto stack
INP	read an integer from keyboard, push onto stack;
OUT	pop top-of-stack and print value plus a newline
NLN	print a (blank) newline
DBG	prints the stack with a newline (useful for debugging)
TRA	toggles trace mode on/off

2.2 Semantics

1. Lines that begin with a percent sign (`%`) are comments and should be ignored.
2. **Errors** immediately halt the machine, **warnings** do not.
3. Instructions should be read from a text file with one instruction per line (the instruction will be the first non-white text on the line; ignore everything after a legal instruction). All irrelevant white space and comment lines should be ignored.
4. SM instructions are case-insensitive (e.g `div` is equivalent to `DIV` and `diV`).
5. The SM operates on integers only.
6. Illegal commands produce warning messages and are then ignored (i.e. thrown away).
   Message format: `[warning - illegal command 'grapefruit' ignored]`
7. The instruction `LDC` requires a single integer argument; if one is not given or it is the wrong type, it is illegal input. Illegal input produces a warning message and is ignored.
   Message format: `[warning - illegal input ignored]`

8. An attempt to divide by zero is an error.
   Message format: `[error - division by zero]`

9. An attempt to access values from an empty stack is an error.
   Message format: `[error - empty stack]`

10. All SM machine input is prompted (with the string '`input: `') and output labeled (with the string '`output: `').

11. The instruction `DBG` prints the entire run-time stack without changing it.
    Message format: `DEBUG:[3,-5,29](top)`, `DEBUG:[](top)`

12. When trace mode is on all file input to the machine is echo printed, along with the line number, to the screen (including blank lines). Initially trace mode is off.
    Message format: `[line #6: % hi mom!]`

13. The instruction `HLT` immediately halts the machine and prints a message.
    Message format: `[machine halted]`

14. Every (error-free) program should stop execution with a `HLT` instruction. Output a warning message if no `HLT` is found. Message format: `[warning - no HLT instruction]`

2.3 Implementation

Your program must execute each command immediately upon reading it from the file (i.e. there is no need to store the entire SM program). Here is the general algorithm:

```
initialization
while there are still instructions and an error hasn't occurred
      and the machine hasn't halted do
  get_next_instruction
  execute_instruction
end while
```

Your program should read the file name from the command line (you may assume the file name is correct and that the file exists). Also be sure to design your program such that you subdivide your code into functions where reasonable. You should include your name in a comment block at the beginning of all uploaded files as well as printing it out first thing when your program executes.

You will find that inputting an entire line at a time is very helpful for this project. However, if you do this you will need to have a means to parse the line and extract the relevant strings. Here is an program that illustrates how one might go about doing this:

```
#include <iostream>
#include <sstream>
using namespace std;

int main (void) {
  string s;

  cout << "Enter a sentence with spaces between words: ";
  getline(cin,s);

  istringstream iss(s);
  cout << "Input: " << s << endl;
  while(true){
    string val;
    iss >> val;
    cout << val << endl;
    if(iss.eof()) break;
  }
  cout << "done!" << endl;
  return 0;
}
```

3 Deliverables

You are to turn in your project (consisting of two files!) through the project submission link on the class web page. Name your source code file `proj3`*XXXX*`.cpp` where *XXXX* is the last 4 digits of your student V number. You should also turn in your class file named `SeqStack`*XXXX*`.cpp` . You must also document your program as mentioned above.

Due date: Tuesday February 27