# Learning Outcomes

Implement a stack and use that stack to implement a postfix notation calculator.

# Instructions

Our last program was to maintain a list of wizards using a Linked List class created by you. In this assignment, you will be writing the stack of a template type which is built upon that Linked List. You'll also be implementing a Postfix expression evaluator class which uses the stack and a driver class.

## Stacks

A stack is an abstract data type which uses a sequential container and limits access to that container to one end. You may enter or remove from the container, but only at one end. Using the Linked List data structure from your last homework assignment, implement a Stack of generic type T.

The Stack should only have one data member: the Linked List. It should implement the following methods:

* Stack(): Empty constructor.
* void clear(): removes all of the elements from the stack
* int size() const: returns the number of elements in the stack
* bool isEmpty() const: returns true if the stack is empty, false otherwise
* T top() const: returns the top of the stack. It must throw an error if the stack is empty.
* T pop(): returns the top of the stack and removes it. It must throw an error if the stack is empty.
* void push(T): adds a value of type T to the stack.

Your linked list should support adding to and from the list from either end. Pick one end and be consistent in your stack implementation. There is no need to write anything other than the above methods for this assignment.

## Postfix Notation

We've been over the rules for Postfix calculation.

* Everything parses from left to right. There's no need to do any backtracking.
* On Token X.
  + If X is a Number: Push X
  + If X is an Operator:
    - Check to see if there are at least two elements on the stack. If not, flag the expression as invalid and quit. Otherwise, keep going.
    - Pop b
    - Pop a
    - Perform calculation c = operator(a, b)
    - Push c
  + If X is a period: Stop evaluating. A period will be our signal to stop.
  + Anything Else: The expression is invalid.

Your postfix calculator should support the addition, subtraction, multiplication, and division operators. You will need a stack of integers to store the numerical data coming into your program. You will also need a boolean flag to keep track of whether the expression is valid or not. Make sure that you set this flag to be true at the beginning of your program. There are two points in the above algorithm which change a flag to false.

Once you are done evaluating, you need to check two things before printing the results: the expression valid flag must be true and there must be exactly one element on the stack. If these are both true, you can print the top of the stack and it should be the correct result. Otherwise, print that the expression is invalid.

## How to Evaluate Numbers and Symbols

Part of this assignment will require you to evaluate the difference between numbers and symbols. For that, I will provide my starting code.

int main() {  
 Stack<int> values;  
 std::string token;  
 bool valid = true;  
  
 std::cout << "Enter a token or '.' to stop: ";  
 std::cin >> token;  
  
 while (token != ".") {  
 int value = std::atoi(token.c\_str());  
  
 if (token == "0" || value != 0) {  
 // Do something with value.  
 }  
 else if (token == "+" || token == "-" || token == "\*" || token == "/") {  
 // Do something with token.  
 }  
 else if (token != "") {  
 // Do something with an invalid token.  
 }  
  
 std::cout << "Enter a token or '.' to stop: ";  
 std::cin >> token;  
 }  
  
 // Evaluate the final part of the expression.

## Write the Driver.

The program will begin with a single prompt: "Enter a token or '.' to stop: "

The user will supply tokens for a postfix expression one at a time. As the tokens are entered, you should evaluate the tokens based on the rules above.

## Example Runs

For example. Run 1.

Enter a token or '.' to stop: 5  
Enter a token or '.' to stop: 2  
Enter a token or '.' to stop: 2  
Enter a token or '.' to stop: +  
Enter a token or '.' to stop: \*  
Enter a token or '.' to stop: .  
The expression is valid.  
Result: 20

For example. Run 2.

Enter a token or '.' to stop: 5  
Enter a token or '.' to stop: 2  
Enter a token or '.' to stop: +  
Enter a token or '.' to stop: 5  
Enter a token or '.' to stop: .  
The expression is invalid.

For example. Run 3.

Enter a token or '.' to stop: 6  
Enter a token or '.' to stop: +  
Enter a token or '.' to stop: .  
The expression is invalid.

For example. Run 4.

Enter a Postfix Expression: 5  
The expression is valid.  
Result: 5

For example. Run 5.

Enter a token or '.' to stop: 100  
Enter a token or '.' to stop: 25  
Enter a token or '.' to stop: /  
Enter a token or '.' to stop: .  
The expression is valid.  
Result: 4

For example. Run 6.

Enter a token or '.' to stop: 10  
Enter a token or '.' to stop: 6  
Enter a token or '.' to stop: -  
Enter a token or '.' to stop: .  
The expression is valid.

Result: 4

# To Hand In

Upload a zip file containing the following files to the drop box on D2L:

* linkedlist.hpp - the header file containing the entire class declaration and external definitions of the linkedlist class.
* stack.hpp - the header file containing the entire class declaration and external definitions of the stack class.
* Your driver with FirstNameLastNameA4.cpp
* Make sure your name, CSCI 3250, and Programming Assignment 4 appear in comments in all of your files.
* Note: NO CREDIT will be given to programming assignments that do not compile.
* Make sure you have compiled and tested your program before handing it in.