# Assignment 4

Part I Due: March 7th, 2014 before 8:30 pm

Part II and III Due : March 14th, 2014 before 8:30 pm

## *Objectives*

- Review command line input and string to integer conversion
- Practice with Exceptions
- Exposure to postfix notation
- Practice using an ADT to solve a problem

## *Introduction*

In this assignment you will implement a program that evaluates expressions written using postfix notation. For example, the result of evaluating `3 7 9 + +` is 19.

You are likely more familiar with expressions written using infix notation such as : `3 + 7 + 9`

The program will accept a postfix expression on the command line and print the result of evaluating the expression using the algorithm below:

```
try
      while there is more input
            if  next token is an operand
                  push value on the stack
            if next token is an operator
                  pop two values from the stack
                  apply the operator to the two values just popped
                  push the result of applying the operator on the stack

      if one element left on stack
            pop value and display it
      else
            invalid expression
catch EmptyStack or NumberFormatException
      invalid expression
```

This assignment has three parts:

1. a connex quiz
2. implement the Stack ADT using a Linked List
3. implement a program that uses a stack to evaluate postfix expressions

You've been provided with a Stack implementation so you can do part 2 or part 3 in any order.

## Part I - Quiz

Follow the “Tests and Quizzes” link on connex and complete the “Assignment 4 Quiz”.

## Part II – Implement the stack interface using a linked list

Create a class called `LLStack` in a file named `LLStack.java`. The class `LLStack` must implement the `Stack` interface specified in `Stack.java` using a linked list structure.

Create an appropriate `Node` class for your linked list implementation in a file named `Node.java`

Modify `StackTester.java` so that it tests your implementation of the `LLStack` class.

Please remember to submit both `LLStack.java` and `Node.java` for Part II.

## Part II – Implement the calculator

Implement a program in a file called `Calc.java` that accepts postfix expressions on the command line and outputs the result of evaluating the expression. Your program must also handle invalid expressions gracefully.

You should break down your Calc program into functions. Solutions that have all the code in the main method will lose marks for poor style.

Your calculator only needs to support integer operands.

Your calculator should support the following binary operators:

| | |
|---|---|
| + | addition |
| - | subtraction |
| / | division |
| x | multiplication |

Note that we are using the lower case letter x to represent multiplication, not the *

The table below shows some of the test cases we will use and the exact output your program must produce for those inputs.

| Command line | Output |
|---|---|
| `java Calc 5 4 +` | `9` |
| `java Calc 5 4 -` | `1` |
| `java Calc 5 4 x` | `20` |
| `java Calc 10 2 /` | `5` |
| `java Calc 1 2 3 4 5 + + + +` | `15` |
| `java Calc 4 +` | `Invalid expression.` |
| `java Calc 4 5 + 6 3 / x` | `18` |
| `java Calc 1 2 + + +` | `Invalid expression.` |
| `java Calc what is this` | `Invalid expression.` |

# Submission

Submit your `LLStack.java, Node.java and Calc.java` using Connex.

**Please be sure you submit your assignment; don't just save a draft**.

A reminder that it is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

We will be using plagiarism detection software on your assignment submissions.

## Grading

| Requirement | Marks |
|---|---|
| Quiz answers | 5 |
| Stack implementation passes test cases | up to 5 |
| Calculator implementation passes test cases | up to 5 |
| Total | 15 |