A5: Expression Evaluator

Overview

You are tasked with parsing an expression given in Reverse Polish Notation (RPN) and converting an expression tree into a stack representing an equivalent expression in RPN.

Deliverables

- **A5.cpp** will be submitted to the A5 code assignment on Autolab.
    - This will contain your code for all required functions. This file should not contain any main function.
    - This is the only code file to be submitted, so please ensure you contain all your code in here.

Due Date

DUE DATE:

A5.cpp: 4/16/2017, at 01:59 (this is Sunday, April 16 at 1:59AM).

Suggested deadline (feel free to ignore):

- **eval should be completed by 4/8/2017 to give sufficient time to work on generate stack.**

Start Early!!!!!!!!

Late Policy

The policy for late submissions is as follows:

- Before the deadline: 100% of what you earn (from your best submission).
- One day late: 50-point penalty (lower your earned points by 50, minimum of 0).
- > 1 days late: 0 points.

Keep track of the time if you are working up until the deadline. Submissions become late after the set deadline.

Objectives

In this assignment, you will create a function to evaluate an expression in RPN and create a function to generate an RPN expression from an expression tree.

Useful Resources

Review the lecture notes and provided examples to get an idea for how trees, stacks, and RPN work. The following Wikipedia articles will help as well:

- https://en.wikipedia.org/wiki/Binary_expression_tree
- https://en.wikipedia.org/wiki/Reverse_Polish_notation

Last updated: Tuesday, Apr 04, 2017 at 12:28 AM

Instructions

- Download the file `A5-skeleton.tar` and extract the skeleton files. **Rename the skeleton files appropriately.**
- A `CMakeLists.txt` file is included to properly build your program. You may use this by Importing the folder into CLion, or from the command line by the following commands:

```
cmake .
make
```

- You will have to add the function definitions in `A5.cpp`
- **You may not use any includes within your A5.cpp file aside from A5.hpp**.

Required:

Complete the functionality for each function. You will lose credit if your code leaks memory.

Evaluate RPN:

```
/**
 * eval
 *
 * @param exprStack: stack holding an expression to evaluate.
 *                Upon completion of evaluating a valid stack,
 *                exprStack should only contain one element, the result
 *                of evaluating the stack.
 *
 * @throw runtime_error: if stack is holding an invalid expression, throw
 *                        a runtime error.
 */
void eval(ItemStack& exprStack) {
    // Your code goes here.

}
```

This function should take a stack holding an expression in reverse Polish notation and evaluate the expression. If the expression is malformed, throw a runtime_error with the message “Malformed expression.” If the expression causes you to divide by 0, throw a runtime_error with the message “DIV by 0 error.”

Process an expression tree:

```
/**
 * generateStack
 *
 * @param root: root of tree holding expression.
 *
 * @return ItemStack: containing the expression represented
 *                    by the tree in RPN (reverse Polish notation)
 *                    with first value on top
 *                    (if reading RPN from left to right)
 */
ItemStack generateStack(ITNode* root);
```

This function should take as input a tree holding an arithmetic expression and convert it into an ItemStack in RPN order with the leftmost element in the RPN expression as the top element of the stack. You do not need to worry about validating the expression. The levels of the tree correspond to the order in which the expressions would be computed (the root node would be the last value to compute).

Bonus:

Create an expression tree:

```
/**
 * generateExprTree
 *
 * @param expr: string containing a possible expression.
 *
 * @return ITNode*: a tree containing the expression tree for the given
 *                  expression expr.
 */
ITNode* generateExprTree (std::string expr);
```

This function should take as input a string with an arithmetic expression and generate the expression tree for the given expression. This function should throw a runtime_error "Invalid expression." if expr contains an invalid expression (unbalanced parenthesis, extra operators or operands, etc.).

Testing

To test your code, you should write tests in the main function in **driver.cpp**. The default code in the main provides some basic testing and an expression to get you started. You must change this to fully test your code.

Submission

Code Submission (100 points):

You must submit your **A5.cpp** file to the A5 assignment on Autolab.

- Your submission must compile and run on Autolab. I suggest that you compile often as you build your code to avoid difficulties debugging syntax and other errors.

- Make sure your code does not leak memory.
- **Bonus (10 points):** Include bonus function in your A5.cpp file as well. We will correctly test it.

Tests will be run on your file and your score will be reported to you when finished. For this assignment there is not limit on the number of submissions you may perform. That said, you should not be using the submission system to test and debug your code.

DUE DATE:

A5.cpp: 4/16/2017, at 01:59 (this is Sunday, April 16 at 1:59AM).

***** Please be aware of this deadline. Autolab will also tell you how long until the deadline.*****