CSE221 Data Structures (Fall 2021)
Instructors: Prof. Young-ri Choi
TAs: Gyeongchan Yun, Jin Yang and Ju Young Bang
Due date: Oct 28, 2021, 11:59 pm.

<u>**Notes**</u>:

- Be aware of plagiarism: You are allowed to use Piazza for QnA, but do **NOT** discuss with your classmates **directly**. If you are confused, ask us by email.
- Check correctness of your code on **uni server** before final submission.
- You are **NOT** allowed to **add STL or other libraries** which means that you cannot add the code such as `#include<stack>` and `#include<queue>`.
- Do **NOT** change the declaration of **given functions**, but you can add any other private/public elements or functions for assignment.
- For grading, we only consider 4 header files in **assignment2/** directory: **`stack.h`**, **`queue.h`**, **`to_postfix.h`** and **`tree.h`**.
- Grading: 20 test cases (5pt for each test case).
- Maximum 3 late days are allowed with a 10% reduction per day.

Assignment 2: Calculator using expression tree

1. Implementing stack and queue (20 pts)

First, you need to implement functions in **`stack.h`** & **`queue.h`**.

You MUST implement stack and queue **using a singly linked list, not an array.** Here is a description of functions you need to implement.

`stack.h`:

- `Stack()` : constructor
- `~Stack()` : destructor
- `void push(const type& item)` : push an element to the stack
- `type& top()` : return the top element in the stack
- `void pop()` : delete the element at the top
- `bool empty()` : return true if the stack does not contain any element
- `int size()` : return the number of elements currently stored

`queue.h`:

- `Queue()` : constructor
- `~Queue()` : destructor
- `void push(const type& item)`: push an element to the queue
- `type& front()` : return the first element in the queue
- `void pop()` : delete the first element in the queue
- `bool  empty()` : return true if the queue does not contain any element
- `int size()` : return the number of elements currently stored

You don’t have to handle pop(), top() and front() on an empty stack/queue.
There should be **NO** upper bound of stack/queue size.

2. Converting infix notation to postfix notation (30 pts)

Second, you need to convert infix notation input to postfix notation.
You can use functions in the `sstream` library [2] which is already included in the given skeleton code of `to_postfix.h.` Refer to the reference for more information on `sstream` library.

You need to implement one function in **`to_postfix.h`**

- `string to_postfix(const string& infix)`

You can assume the following for the **input expression**:

1. There are **only five operators**, i.e., +, – (binary minus), *, /, and unary minus. Make sure that unary minus operator works correctly. To do so, you need to come up with rules to distinguish unary and binary operators, covering different cases such as:

–10 + 3 : unary minus
10 – 2 : binary minus
(10 + 5) – 8 : binary minus
(-10 * 5) + 2 : unary minus

2. There may be parentheses in the expression, but the expression may **NOT** be fully parenthesized. For example, (2+3*5)+7 can be a valid input, although the fully-parenthesized expression will be ((2+(3*5))+7).

3. Input strings may contain numbers, parenthesis, operators, and space **only**. Example) 10 + { 20 + 30 } is not allowed because { and } are not allowed to use.
4. You can assume that there will be **no grammatical errors** in the given infix expression. For example, 10 (20+30) is error because an operator is missing between 10 and (.
5. When input is empty string (`""`) or space(s) (`"   "`), it should return empty string (`""`).

Output format
All operators and operands, except unary minus, should be separated with single space. Unary minus characters should have no spaces with its trailing operand. There will be no parentheses in the postfix notation.
Example)

Infix: "(7+4)*2–1"
Return: "7 4 + 2 * 1 –"

Infix: "–10 + 3 – 2"
Return: "–10 3 + 2 –"

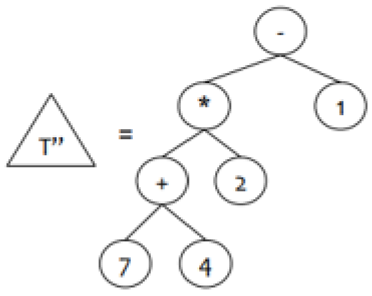
3. Evaluating expression tree (50 pts)

Third, we can make a calculator evaluating the expression tree.
You can use functions in the `cstdlib` [1] library and `string(sstream)` [2] library which are already included in the given skeleton code of `tree.h`. Please refer to the references.

You need to implement functions in **`tree.h`**

- `Tree* build_expression_tree(const string& postfix)`
  Build an expression tree using **Stack** in `stack.h`.
  The expression tree is a binary tree in which each internal node corresponds to the operator and each leaf node corresponds to the operand.

  Example) Given the postfix notation: 7 4 + 2 * 1 –, an expression tree is built as follows:



- `void Tree::print()`
  Print out nodes in the expression tree in **level order. "null"** should be included to signify a path terminator where no node exists below **except** nulls that come after the nodes in the last level.
  Note that you MUST implement it using **Queue** in `queue.h`.

  Example)
  Given the expression tree for the postfix notation: 7 4 + 2 * 1 –, output of `print()` is as follows:

  (–,*,1,+,2,null,null,7,4)

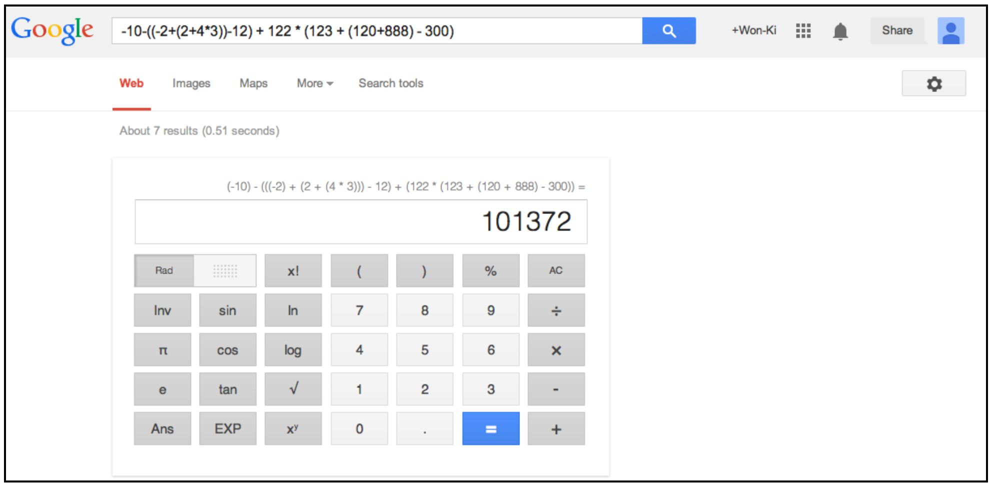
  Given the expression tree for the postfix notation: 1 2 + 3 4 5 + * *, output of `print()` is as follows:

  (*,+,*,1,2,3,+,null,null,null,null,null,null,4,5)

  Output format
  Notice that there is no **whitespace,** and it separates a **new line**.

- `double Tree::eval()`
  Evaluate operands with operators in the expression tree.
  The function returns the calculated value in `double` format. It should be able to take both **integer-based** expression and **float-based** expression as input.

Once you finish implementation, check the correctness of your code by comparing the solutions using google (if you copy-and-paste the expression into google search window then google will give you the solution. See below.)



```
-10-((-2+(2+4*3))-12) + 122 * (123 + (120+888) - 300)
```

About 7 results (0.51 seconds)

```
(-10) - (((-2) + (2 + (4 * 3))) - 12) + (122 * (123 + (120 + 888) - 300)) =
101372
```

Compile/Execution Method

To compile the given files, enter the following command.

$ g++ -o assign2 main.cpp stack.h queue.h to_postfix.h tree.h

When the compilation is done, the following command will execute your code.

$ ./assign2

Submission Method

Note that main.cpp is an example of testcase which can be modified by yourself to check various cases. We only consider 4 header files in **assignment2/** directory for grading:

`stack.h`, **`queue.h`**, **`to_postfix.h`** and **`tree.h`**.

This assignment should be submitted through GitLab. You will be able to push your source code through the following commands.

$ cd uni+student_id/assignment2
$ git add stack.h queue.h to_postfix.h tree.h
$ git commit -m "Final commit" (commit message is not important for grading)
$ git push origin master

References

[1] https://www.cplusplus.com/reference/cstdlib/
[2] https://en.cppreference.com/w/cpp/string/basic_string