Data Structures and Algorithms Spring 2019
Assignment 2

Due: 2/27/2019 on Canvas
(100 points)

Instructions Submit your answers and code in Canvas. Include a brief README file explaining your code, especially if you implemented some of the suggestions for extra credit.

Question 1 (15 points)

Write a program `void reverse_list(list **l)` in pseudo-code or C++ to reverse the direction of a given singly-linked list. In other words, after the reversal all pointers should now point backwards. Your algorithm should take linear time. The node of this singly-linked list is defined as

```
typedef struct list {
  item_type item;
  struct list * next ;
}list;
```

```
void reverse_list(list **l){









}
```

Remark: 15 points for completely correct. 10 points for correctly reversing the list with minor errors.

Question 2 (85 points)

Implement a stack and solutions to the following problems: balancing parenthesis, evaluating postfix expressions and transforming infix expressions into postfix expressions.

We are providing some sample code and input files:

public/
- balancing.cpp
  - main method to check whether an expression is balanced
- infix2postfix.cpp
  - main method to transform an infix expression into postfix
- input_balanced.txt
  - test cases for balancing.cpp
- input_infix2postfix.txt
  - test cases for infixtopostfix.cpp
- input_postfixEval.txt
  - test cases for postfixEval.cpp
- postfixEval.cpp
  - main method to evaluate postfix expressions
- stack.cpp

- stack implementation
stack.hpp
- stack header file

• To compile, run

$ g++ -o stack.cpp balancing.cpp

$ g++ -o stack.cpp postfixEval.cpp

$ g++ -o stack.cpp infixtopostfix.cpp

• To run each program, run

$ ./a.out

• The test cases follow this format: expected_solution input. Given input, your job is to implement code that gets the expected_solution. Obviously, you need to calculate expected_solution, not just print it.

• balancing.cpp must balance round parentheses, and square and curly brackets (() [] {})

• While we provide a few test cases, you are expected to add more to make sure your code works.

Grading:

Implementing the stack is worth 15 points. Solving the balancing problem is worth 20 points, evaluating postfix expressions is worth 20 points, and transforming infix expressions into postfix is worth another 30 points.

You also have a few opportunities to earn extra credit:

• Transform postfix expressions into infix expressions.

• Evaluate postfix expressions when the operands can be any number (not only one digit). The easiest way to do this is to use whitespace as a delimiter.

• Transform infix expressions into postfix, but also allow for the sign operator ('−'). The code we say in class assumes that all operators are binary, but the sign operator is unary, e.g., −(2∗4).