CS2710 - Programming and Data Structures Lab

Lab 6 (graded)

Sept 4, 2024

Instructions

- You are expected to solve ALL the problems in the lab, using the local computer, C++ language, and g++ compiler. (Bonus problems can fetch 5% capped bonus, and Optional problems are just for fun and won't be graded.)
- You should submit your code to the course **moodle** on time, i.e., on/before 4.45pm (so that TAs can subsequently grade your submissions for graded lab assignments using the private test cases).
- You must strictly adhere to the following naming convention for your .cpp files and the single .zip file submission. For example, for a Lab Session 6 consisting of 2 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_LAB6_Q1.cpp** and **CS23B000_LAB6_Q2.cpp**
  - Put both these .cpp files in a directory named **CS23B000_LAB6**
  - Zip this directory into a file named **CS23B000_LAB6.zip**
  - Submit only this single .zip file to moodle.
- The questions are based on your training in programming in CS1111, concepts seen in class in CS2700, and changes needed to switch from C to C++.
- If you need assistance, ask your TA, not your classmate.
- Internet access and mobile devices are prohibited in the lab.
- Check course Moodle for the public test cases and the evaluation script, which you can use to test your programs.
- Solve each problem in this lab session using *linked list or stack* as the primary data structure (do not use any other data structure; string data type is fine - please clarify with the TAs if you have any questions about this).

---

1. [ONE LINK TO RULE THEM ALL AND BIND THEM!] Write a C++ class to implement a singly linked list (SLL) supporting the following operations:

   - Create a new list (specified via the command "NEW").
   - Insert a new node at the beginning or end of the list ("PUSH_BEGIN x" or "PUSH_END y").
   - Delete the first occurence of node with value z (if it exists). ("DELETE z").

   Input Format:

   - N number of commands

   Output Format:

   - Print the contents of the list (as space-separated values)

   Constraints:
   $2 \leq N \leq 20$
   Only valid commands will be given

   Examples:
   Input1:
   5
   NEW PUSH_BEGIN 5 PUSH_END 6 PUSH_BEGIN 8 DELETE 6
   Output1:
   8 5
   Explanation:
   1 command : make a dummy head
   2 command : push node with value 5 in the beginning of the list : head->5->NULL
   3 command : push node with value 6 at the end of the list : head->5->6->NULL
   4 command : push node with value 8 at the beginning of the list : head->8->5->6->NULL
   5 command : search the list for value 6 and delete it : head->8->5->NULL

   Input2:
   4
   NEW PUSH_END 7 PUSH_END 6 PUSH_END 8
   Output2:
   7 6 8
   Explanation:
   1 command : make a dummy head
   2 command : push node with value 7 at the end of the list : head->7->NULL
   3 command : push node with value 6 at the end of the list : head->7->6->NULL
   4 command : push node with value 8 at the end of the list : head->7->6->8->NULL

2. [AND A LINK TO FURTHER TWIST THEM!] To your implementation in the Question 1 above, add support for two extra operations/commands given below. Use only pointer-based manipulations to implement these operations - specifically do not change the data/value fields of the list nodes when implementing these operations.

   - Reverse the order of the first k nodes in the list, without changing the order of the remaining nodes ("REVERSE k").
   - Rotate the list towards right by x units ("ROTATE x").

   Input Format:

   - N number of commands
   - Commands with valid values separated by space

   Output Format:

   - Print the contents of the list (as space-separated values)

   Constraints:
   $2 \leq N \leq 20$

   Examples:
   Input1:
   7
   NEW PUSH_BEGIN 5 PUSH_BEGIN 6 PUSH_BEGIN 8 PUSH_BEGIN 9 PUSH_BEGIN 14 REVERSE 3
   Output1:
   8 9 14 6 5
   Explanation:
   1 command : make a dummy head
   2 command : push node with value 5 in the beginning of the list : head->5->NULL
   3 command : push node with value 6 in the beginning of the list : head->6->5->NULL
   4 command : push node with value 8 at the beginning of the list : head->8->6->5->NULL
   5 command : push node with value 9 at the beginning of the list : head->9->8->6->5->NULL
   6 command : push node with value 14 at the beginning of the list : head->14->9->8->6->5->NULL
   7 command : reverse the order of the first 3 nodes : head->8->9->14->6->5->NULL

   Input2:
   6
   NEW PUSH_END 10 PUSH_END 5 PUSH_END 1 PUSH_BEGIN 99 ROTATE 2
   Output2:
   5 1 99 10
   Explanation:
   1 command : make a dummy head
   2 command : push node with value 10 at the end of the list : head->10->NULL
   3 command : push node with value 5 at the end of the list : head->10->5->NULL
   4 command : push node with value 1 at the end of the list : head->10->5->1->NULL
   5 command : push node with value 99 in the beginning of the list : head->99->10->5->1->NULL
   6 command : rotate towards right by 2 units : head->5->1->99->10->NULL

3. [HOW DEEP CAN ONE GO WITHOUT LOSING FORM?] Given a string P of parantheses of different types ("{, }, (,), [,]"), write a program to check if P is well-formed. We call P as well-formed if it is balanced and nested, as seen in class. Your program should not only output whether P is well-formed, but also the maximum level of nesting for well-formed P. You can use stacks (e.g., std::stack) to solve it.

Input Format:

- N (# of characters in P)
- P (string of parantheses)

Output Format:

- True or False (True if P is well-formed and False otherwise)
- Maximum nesting depth (only if P is well-formed)

Constraints:

- $1 \leq N \leq 100$

Examples:
Input1:
16
$[()]\{\}\{[()()]()\}$
Output1:
True
3
Explanation:
The parantheses are well-formed, i.e., properly balanced and nested. The maximum nesting depth is 3, and follows from the nesting depth shown as integers at various points along this string: $[(2)]\{1\}\{[(3)(3)](2)\}$.

Input2:
4
$[(])$
Output2:
False
Explanation:
1 and 4 brackets are not nested because there is a closing ']' before the closing paranthesis for '('. Nothing is reported for nesting depth, since the string is not well-formed.

4. [**BONUS QUESTION** - MONOTONIC STACK] You are given a non-negative number as a string of N digits and an integer k. Your task is to return the smallest possible number that can be obtained by removing exactly k digits from the input string of N digits. You can use stacks (e.g., std::stack) to solve it.

   Input Format:

   - k (# of digits to remove)
   - N (length of string)
   - N-digit number (represented as a string)

   Output Format:

   - Output the final smallest number.

   Constraints:

   - $1 \leq k \leq N \leq 1000$

   Examples:
   Input1:
   3
   7
   1432219
   Output1:
   1219
   Explanation:
   Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.

   Input2:
   1
   5
   10200
   Output2:
   200
   Explanation:
   Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.