CS2100 Computer Programming Lab

Date : August 31, 2015
First submission deadline: 6th September (problem 1)
Second submission deadline: 13th September (problems 2 & 3)

Instructions

This exercise consists of 3 problems which you need to solve as an assignment. The first problem is a stack library which will be used in the remaining two problems as well. Please go through all the problems carefully before you start coding.

Submission Guidelines

The problem 1 needs to be submitted on 6th of September. The other two problems need to be submitted on 13th of September. In order to submit the solutions, create a folder named `cs14bXXX_lab4_1`. You need to include in this folder your code for the problem 1. Name your program as `cs14bXXX_p1.c`. Follow similar guidelines for lab 4.2 submission. Include problem 2 and problem 3 in the folder `cs14bXXX_lab4_2`. Compress these folders and submit a tar ball that is named `cs14bXXX_lab4_1.tar.gz` and `cs14bXXX_lab4_2.tar.gz` respectively. Please make sure you have removed all the temporary files and executables before you make your submission on Moodle.

Problems

1. **Problem 1.** Write a C program to implement a standard **Stack library** using **linked-lists**. The library should provide standard stack operations, i.e, **Push()**, **Pop()**, **IsEmpty()**, and **Top()**. Your stack should contain entries of the type **struct SNode**. Your functions should have following signatures:

   - **void Push (struct SNode** top, char ch)**

     `\\This function pushes character ch to the stack.`
   - **char Pop (struct SNode** top)**

     `\\This function pops top character from the stack. Assume stack is not empty.`
   - **bool IsEmpty(struct SNode* top)**

     `\\This function checks whether stack is empty.`
   - **char Top (struct SNode* top)**

     `\\This function returns top element of stack. Assume stack is not empty.`

   Your program should add following code snippet with global scope

```
struct SNode
{
    char data;
    struct SNode* next;
};
```

2. **Problem 2.** In this exercise we are interested in checking whether a given string is well-parenthesized or not. Such a function is useful when you are interested in evaluating mathematical formula. However, for our exercise we will assume that the input consists of only parenthesis of 3 possible forms (), [], {}. Given an input string containing a sequence of parenthesis, write a function that returns **true** if the given sequence is a balanced parenthesis else return **false**. Input string consists of following symbols: `{`, `}`, `(`, `)`, `[`, `]`. Each opening parenthesis matches only with the same type of closing parenthesis. For example { }, `( )` and so on.

   Your function should have following signature:

   - **bool IsBalanceParanthesis (char* s)**

   Example

   ```
   Input: {{([])}}          Output: 1
   Input: ([)]{}            Output: 0
   ```

3. **Problem 3.** Consider a man starts walking from a point $(0, 0)$ on the co-ordinate system. The man can make only one of the following moves at a time: he moves Up, goes Down, goes Left, or goes Right, he can backtrack a few steps. We are given a sequence of moves made by the man. Our goal is to find the final position of the man. The sequence of moves of a man are given as (U,D,L,R,Bk) where U implies up, D implies down, L implies left, R implies right, and Bk is backtrack previous k steps. Each movement is of unit length. Assuming the man is facing the page, navigation for input string RURDDRUUB2UU is shown in Figure 1. We will use the following structure to represent the current position of the man:

   ```
   struct position
   {
       int x;
       int y;
   };
   ```

   Write a function **FinalPosition** which returns the final position of the man after following the set of moves starting at $(0, 0)$. Your function should have following signature:

   - **struct position FinalPosition (char* s)**

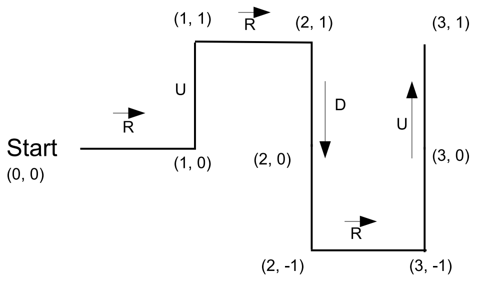

Figure 1: Path navigation

Sample Testcases

```
Input: LLLUUB2RU          Output: -2 1
Input: UURDLLDLB3RR       Output:  2 1
```