CS 32 Week 4 Worksheet

This worksheet is entirely **optional**, and meant for extra practice. Some problems will be more challenging than others and are designed to have you apply your knowledge beyond the examples presented in lecture, discussion or projects. All exams will be done on paper, so it is in your best interest to practice these problems by hand and not rely on a compiler.

If you have any questions or concerns please email raykwan@ucla.edu, or go to any of the LA office hours.

Concepts

Stacks, Queues

1) Given a string of '(', ')', '[', and ']', write a function to check if the input string is *valid*. Validity is determined by each '(' having a corresponding ')', and each '[' having a corresponding ']', with parentheses being properly nested and brackets being properly nested
   Examples: "[()[[([][])]]]" → Valid
   "((([[]])))" → Invalid
   "(()))" → Invalid
   "()[]" → Valid

```
bool isValid(string parens) {
      // Fill in code here
}
```

2) Give an algorithm for reversing a queue Q. Only following standard operations are allowed on queue:
   a) Q.push(x) : Add an item x to the back of the queue.
   b) Q.pop() : Remove an item from the front of the queue.
   c) Q.front() : Return the item at the front of the queue
   d) Q.empty() : Checks if the queue is empty or not.

   You may use an additional data structure if you wish.

   Example:
   Input : Q = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
   Output :Q = [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]

```
void reverseQueue(queue<int>& Q){
      // your code goes here
}
```

3) Implement a Stack class using only queues as data structures. This class should implement the *empty*, *size*, *top*, *push*, and *pop* member functions, as specified by the standard library's implementation of stack. (The implementation will not be very efficient.)

4) Implement a Queue class using only stacks as data structures. This class should implement the *empty*, *size*, *front*, *back*, *push*, and *pop* member functions, as specified by the standard library's implementation of queue. (The implementation will not be very efficient.)

5) Write a function *findNextInts* that takes in two integer arrays of size *n: sequence* and *results*. This function assumes that *sequence* already contains a sequence of positive integers. For each position *i* (from 0 to *n*-1) of *sequence*, this function should find the smallest index *j* such that *j > i* and *sequence[j] > sequence[i]*, and put *sequence[j]* in *results[i]*; if there is no such *j*, put -1 in *sequence[i]*. Try to do this without nested for loops both iterating over the array! (Hint: `#include <stack>`).

```
void findNextInts(const int sequence[], int results[], int n);
```

Example:

```
int seq[] = {2, 6, 3, 1, 9, 4, 7 }; // Only positive integers!
int res[7];
findNextInts(seq, res, 7);
for (int i = 0; i < 7; i++) { // Should print: 6 9 9 9 -1 7 -1
  cout << res[i] << " ";
}
cout << endl;
```

Notice that the last value in *results* will always be set to -1 since there are no integers in *sequence* after the last one!

6) Evaluate the following postfix expression and show your work:
9 5 * 8 - 6 7 * 5 3 - / *