Data Structures and Algorithms

Lab 2 (5% of final grade)

(100 points total)

All code must be your own and follow the guidelines in lecture notes and textbook. Use proper coding styles including comments and indentation. No code from Internet sources is allowed in this assignment. All code must be in C++ programming language.

1. Implement the Stack-based Parentheses Matching algorithm using various implementation versions of Stack:
    a. built-in STL class Stack
    b. array-based Stack
    c. linked-list based Stack

   Allow the user to select either of the above Stack implementations. Test your implementation using inputs (( ))(( )){([( )])} and ({[ ])}. Submit screenshots of the results and code files in a zip file.

2. Design an ADT to store the team members of a team composed of two subteams – one Varsity and the other JV. This ADT consists of two stacks—one “V” and one “JV”—and has as its operations coded versions of the regular stack ADT operations based on the players of which subteam we add. For example, this ADT should allow for both a “V” push operation and a “JV” push operation. Give an efficient implementation of this ADT using a single array whose capacity is set at some value *N* that is assumed to always be larger than the sizes of the “V” and “JV” subteams combined. Create and test the algorithm with a realistic input. Submit a writeup of your design, screenshots of the results, and code files in a zip file.

3. Implement the simulation of access to a shared printer by computers on a network by using various implementation versions of Queue.

    a. built-in STL class Queue
    b. array-based Queue
    c. circular linked-list based Queue

   Allow the user to select either of the above Queue implementations. Create and test the algorithm with a realistic input. Submit screenshots of the results and code files in a zip file.

4. Implement the capital gains algorithm solving the problem below using various implementation versions of Queue. When a share of common stock of some company is sold, the capital gain (or, sometimes, loss) is the difference between the share's selling price and the price originally paid to buy it. This rule is easy to under stand for a single share, but if we sell multiple shares of stock bought over a long period of time, then we must identify the shares actually being sold. A standard accounting principle for identifying which shares of a stock were sold in such a case is to use a FIFO protocol—the shares sold are the ones that have been held the longest (indeed, this is the default method built into several personal finance software packages). For example, suppose we buy 100 shares at \$20 each on day 1, 20 shares at \$24 on day 2, 200 shares at \$36 on day 3, and then sell 150 shares on day 4 at \$30 each. Then applying the FIFO protocol means that of the 150 shares sold, 100 were bought on day 1, 20 were bought on day 2, and 30 were bought on day 3. The capital gain in this case would therefore be $100 \cdot 10 + 20 \cdot 6 + 30 \cdot (-6)$, or \$940. Write a program that takes as input a sequence of transactions of the form "buy x share(s) at \$y each" or "sell x share(s) at \$y each," assuming that the transactions occur on consecutive days and the values x and y are integers. Given this input sequence, the output should be the total capital gain (or loss) for the entire sequence, using the FIFO protocol to identify shares.

   a. built-in STL class Queue
   b. array-based Queue
   c. circular linked-list based Queue

   Allow the user to select either of the above Queue implementations. Create and test the algorithm with a realistic input. Submit screenshots of the results and code files in a zip file.

5. Simulate the process of a card dealer who can deal off both the top and the bottom of the card set. Implement the process using various implementation versions of Deque:
   a. built-in C++ STL class Deque
   b. circular array-based Deque
   c. doubly linked-list based Deque

   Allow the user to select either of the above Deque implementations. Create and test the algorithm with a realistic input. Submit screenshots of the results and code files in a zip file.