# Sample Solution for CS3323 Fall 2006 Assignment 2 **(37 marks)**

Due Monday, Oct. 16, by 5pm.

---

- Assignments should be handed in by placing them in the CS3323 bin on E level of Gillin Hall.
- A hardcopy of your program code must be submitted with your written assignment, code must also be submitted electronically by email to hzhang@unb.ca as an attachment. The name of your attachment must consist of your last name, student id, assignment number, question number. For example, your last name is Smith, student id is: 330321, it is the code for Question 6 in Assignment 2, then the name of your attachment should be "Smith-330321-2-6.java".

---

1. **(5 marks)** Describe the output of the following series of stack operations on a single, initially empty stack:
   *push*(5), *push*(3), *pop*(), *push*(2), *push*(8), *pop*(), *pop*(), *push*(9), *push*(1), *pop*(), *push*(7), *push*(6), *pop*(), *pop*(), *push*(4), *pop*(), *pop*().

   **solution:** (bottom of the stack is to the left)
   5
   5 3
   5
   5 2
   5 2 8
   5 2
   5
   5 9
   5 9 1
   5 9
   5 9 7
   5 9 7 6
   5 9 7
   5 9
   5 9 4
   5 9

5

2. **(5 marks)** Describe the output of the following series of queue operations on a single, initially empty queue:
*enqueue*(5), *enqueue*(3), *dequeue*(), *enqueue*(2), *enqueue*(8), *dequeue*(), *dequeue*(), *enqueue*(9), *enqueue*(1), *dequeue*(), *enqueue*(7), *enqueue*(6), *dequeue*(), *dequeue*(), *enqueue*(4), *dequeue*(), *dequeue*().
**solution:** The head of the queue is on the left.
5
5 3
3
3 2
3 2 8
2 8
8
8 9
8 9 1
9 1
9 1 7
9 1 7 6
1 7 6
7 6
7 6 4
6 4
4

3. **(5 marks)** Describe in pseudo-code a linear-time algorithm for reversing a queue $Q$. To access the queue, you are only allowed to use the methods of queue ADT. *Hint*: Consider using an auxiliary data structure.

**solution:** We empty queue $Q$ into an initially empty stack $S$, and then empty $S$ back into $Q$.

**Algorithm** ReverseQueue($Q$)
Input: queue $Q$
Output: queue $Q$ in reverse order
$S$ is an empty stack

**while** (! $Q$.isEmpty()) **do**
    $S$.push($Q$.dequeue())
**while** (! $S$.isEmpty()) **do**
    $Q$.enqueue($S$.pop())

4. **(10 marks)** Describe how to implement two stacks using one array. The total number of elements in both stacks is limited by the array length; all stack operations should run in $O(1)$ time.

   **solution:** Let us make the stacks ($S_1$ and $S_2$) grow from the beginning and the end of the array ($A$) in opposite directions.

   Let the indices $T_1$ and $T_2$ represent the tops of $S_1$ and $S_2$ correspondingly. $S_1$ occupies places $A[0 \ldots T_1]$, while $S_2$ occupies places $A[T_2 \ldots (n-1)]$. The size of $S_1$ is $T_1 + 1$; the size of $S_2$ is $n - T_2 + 1$. Stack $S_1$ grows right while stack $S_2$ grows left. Then we can perform all the stack operations in constant time similarly to how it is done in the basic array implementation of stack except for some modifications to account for the fact that the second stack is growing in a different direction. Also to check whether any one of the stacks is full, we check if $S_1$.size() + $S_2$.size()$\geq n$. In other words, the stacks do not overlap if their total length does not exceed $n$.

5. **(5 marks)** Describe in pseudo-code a linear time - $O(n)$ - algorithm which copies the elements of array A into a new array B in such a way that B contains all the elements of A with any odd integers located before any that are even. For example,

   2 3 1 6 8 9 4 7 5 10 $\leftarrow$ A

   becomes something similar to:

   3 1 9 7 5 2 6 8 4 10 $\leftarrow$ B

   Hint: The order of the integers in each grouping (odd/even) in B need not be preserved.

   **solution:**

```
Algorithm oddBeforeEven(theInputQueue)
        Treat the array as a queue, call it theInputQueue
        returns theOutputQueue with odds before evens
        Create an auxiliary data structure theStack
        while (! theInputQueue.isEmpty()) do {
            a ← theInputQueue.dequeue()
            if ( a is odd ) then {
              theOutputQueue.enqueue(a)
```

```
        }
        else {
          theStack.push(a)
        }
    }
    while (! theStack.isEmpty()) do {
        theOutputQueue.enqueue(theStack.pop())
    }
```

6. **(12 marks)** Write a Java program to calculate postfix expressions using a stack, and convert postfix to infix. The input to your program will be text, consisting of one expression per line. All expressions will consist of numbers and operators from the operator set {+, -, *, /}, separated by spaces. For each expression, compute both the infix equivalent of the expression and its value, and print them as an equation. It is acceptable to add parentheses even if they are not needed.

   If an input expression is not a correctly formed postfix expression (eg., 5+6), an error message should be displayed.

   To simplify reading the input, all numbers in the input expressions will be nonnegative integers. This will not necessarily be true of intermediate values of the output.

   Example input:
   22 6 5 + / 9 -
   10 7 - 17 8 1 + - *
   8 5 / 3 +
   5 + 6

   Example output:
   22 / (6 + 5) - 9 = -7
   (10 - 7) * (17 - (8 + 1)) = 24
   8 / 5 + 3 = 46
   Not a valid postfix expression.

   **Marking Scheme:**

   - Functionality
     - postfix evaluation (4 marks)
     - postfix → infix conversion (4 marks)

- 8 test cases (4 marks - 0.5 per test case)