# Data Structures and Algorithms

# INFO 6205

# Saturday: Homework 2

# Due: January 27, 2019

Put all your java, compiled class files and documentation files into a zip file named Homework2.zip and submit it via the drop box on the blackboard before the END of due date. Put your name on all .java files. There will be a short quiz on this homework.

1. Human use Infix expression and computers use Postfix expression. Consider these

Infix expressions:

(1 + 2 \* (20 / 5 ))

(1 + 3 + ( ( 4 / 2 ) \* ( 8 \* 4 ) ))

(4 + 8) \* (6 - 5)/((3 - 2) \* (2 + 2))

a) Evaluate Infix expression

i) use two arrays

ii) use single array

iii) discuss the running time of (i) and (ii) this algorithm runs in linear time

b) Convert Infix expression to Postfix

c) Evaluate Postfix expression

Write Java code for (a,b,c) as described above, compile and run Infix examples.

2. Consider the following Algorithm to convert Infix expression to Postfix.

a) Consider Infix expression example: ( ( A / ( B + C ) ) - D )

b) Apply Algorithm to Infix example, show step-by-step

c) Write Java code for the algorithm to convert Infix to Postfix expression

Algorithm:

while there are more symbols to read

read the next symbol

case:

operand --> output it.

’(’ --> push it on the stack.

’)’ --> pop operators from the stack to the output

until a ’(’ is popped; do not output either of

the parentheses.

operator --> pop higher- or equal-precedence operators

from the stack to the output; stop before

popping a lower-precedence operator or

a ’(’. Push the operator on the stack.

end case

end while

pop the remaining operators from the stack to the output

Algorithm:

while there are more symbols to read

read the next symbol

case:

operand --> output it.

’(’ --> push it on the stack.

’(’ --> push it on the stack.

’A’ --> push it on the stack.

’/’ --> push it on the stack.

’(’ --> push it on the stack.

’B’ --> push it on the stack.

’+’ --> because ‘+’its priority less then ‘/’, so push it on the stack.

’C’ --> push it on the stack.

’)’ --> pop operators from the stack to the output

until a ’(’ is popped; do not output either of

the parentheses. For now pop() ‘/’.

’-’ --> For now there is no any operator in the stack. So push it on

the stack.

’D’ --> push it on the stack.

’)’ --> pop operators from the stack to the output

until a ’(’ is popped; do not output either of

the parentheses.

operator --> pop higher- or equal-precedence operators

from the stack to the output; stop before

popping a lower-precedence operator or

a ’(’. Push the operator on the stack.

end case

end while

pop the remaining operators from the stack to the output

the result should be ‘ABC+/D-’

3. For the LinkedList implementation of Queue example discussed in class, write a TestLinkedListQueue class to test enqueue, dequeue,, isEmpty and other operations as needed with

{item1, item2, item3, item4, item5}

4. Describe the Array Implementation of Queue with the following example. Use ample main() shown below and write the Java code for enqueue and dequeue, and other operations as necessary and manage the head and tail pointers. Test and Print the queue at each operation.

**public** **static** **void** main(String a[]){

        MyQueue queue = **new** MyQueue(5);

        queue.enqueue(4);

        queue.dequeue();

        queue.enqueue(56);

        queue.enqueue(2);

        queue.enqueue(67);

        queue.dequeue();

        queue.dequeue();

5.Write a Iterative method to sumDigits that has one integer parameter and returns the sum of the digits in the integer specified. The method should throw an IllegalArgumentException if the integer specified is negative. Remember, your method should not use iterative loops. For example, if the integer is 26397, then this method should return 27.

6. Write a recursive method countStringBinary that has one integer parameter n and returns the number of binary strings of length n that do not have two consecutive 0's. For example, for n = 4, the number of binary strings of length 4 that do not contain two consecutive 0's is 8: 1111, 1110, 1101, 1011, 1010, 0111, 0110, 0101