Data Structures and Algorithms

INFO 6205

Homework 3

Due: October 3, 2020

Put all your java, compiled class files and documentation files into a zip file named Homework3.zip and submit it via the drop box on Canvas before the END of due date.

Put your name on all .java files. There will be a short quiz on this homework.

1. For the LinkedList implementation of Queue example “to be or not to be” I discussed in class, write a TestLinkedListQueue class to test enqueue, dequeue,, isEmpty and other operations as needed.

2. Describe the Array Implementation of Queue with “It was the best of times” example. You need to provide a sample data and walk through the enqueue and dequeue, and other operations as necessary and manage the head and tail pointers.

3. Consider the following QueueOfStrings code to manage queue. The input to this method is String “The temperature - - degrees today and it - - - tomorrow”.

A) Show step-by-step of queue execution

B) What is the output

public static void main(String[] args)  
 {  
 QueueOfStrings q = new QueueOfStrings();  
 while (!StdIn.isEmpty())

{

String s = StdIn.readString();

if (s.equals("-")) StdOut.print(q.dequeue());

else q.enqueue(s);

}   
}

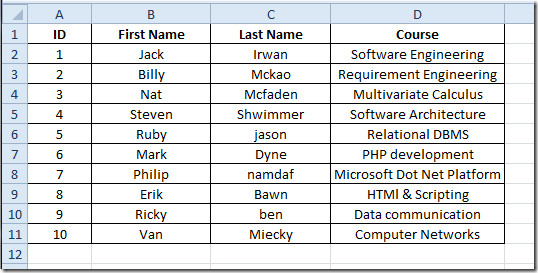
4. Consider this Algorithm to “Evaluate Postfix Expression”: 10 2 8 \* + 3 -

Algorithm: Maintain a stack and scan the postfix expression from left to right – When we get a number, output it – When we get an operator, pop the top element in the stack until there is no operator having higher priority then this operator, and then push (operator) into the stack – When the expression is ended, pop all the operators remain in the stack:

A) Show Stack step-by-step

B) Write Java code to compute postfix expression

5. Consider the following data:



Build Queue with LinkedList implementation and Array implementation:

a) Create file “input.txt” with this data

b) Read input.data into an an ArrayList.

c) Create Queue with LinkedList implementation

d) Write Node data structure of your input data

e) Queue must support all operations of queue

f) Write a Test program to test your linked implementation of Queue:

—enqueue all elements into queue

—dequeue 4 elements from queue

—enqueue all elements into queue

—dequeue all elements from queue

—dequeue 1 element

—enqueue all elements into queue

—enqueue this element into the queue:

11 John Johnson Java Programming

—Print queue with the goal:

i) reverse order ii) original order as was first read into array list

g) Compile and Run your program

h) what is Queue LinkedList time-complexity?

i) Repeat (a)—(g) with Queue fixed Array Implementation

j) what is Queue Fixed Array time-complexity?

k) What are the consequences of oversizing or undersizing fixed array size?

6. Consider following Algorithm to “Evaluate Infix Expressions” with Two arrays:

Test data:

(A + B) \* C + D / (E + F \* G) - H

(300 + 23) \* (43 - 21) / (84 + 7)

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

A) Step through algorithm to develop a Stack Table for for each Infix expression

B) Write Java code to test each Infix Expression

C) Compile and Run

Algorithm:

Iterate through given expression, one character at a time

1. If the character is an operand, push it to the operand stack.

2. If the character is an operator,

1. If the operator stack is empty then push it to the operator stack.

2. Else If the operator stack is not empty,

• If the character’s precedence is greater than or equal to the precedence of the stack top of the operator stack, then push the character to the operator stack.

• If the character’s precedence is less than the precedence of the stack top of the operator stack then do Process (as explained above) until character’s precedence is less or stack is not empty.

3. If the character is “(“, then push it onto the operator stack.

4. If the character is “)”, then do Process (as explained above) until the corresponding “(” is encountered in operator stack. Now just pop out the “(“.

Once the expression iteration is completed and the operator stack is not empty,

do Process until the operator stack is empty.  The values left in the operand stack

is our final result.

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

A) Infix expression example: A \* B / C + (D + E - (F \* (G / H)))

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 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