# Data Structures and Algorithms

# INFO 6205

# Homework4

# Due: March 7, 2022

Put all your java, compiled class files and documentation files into a zip file Homework3.zip

and submit it via drop box on Canvas before the END of due date. Put your name on all .java files.

1. Consider the following:

a) For Power (x, n) with n=7, a) write Iterative algorithm step-by-step, and b) write Recursive algorithm, step-by-step, c) Write Java code for a and b, compile and run

b) For Towers of Hanoi problem with n=8 discs, how does the algorithm work? What data structures would you use? provide step by step operations. Write Java code, compile and run program. <https://introcs.cs.princeton.edu/java/23recursion/>

2. For the following Algorithm

a) Why would you use Grey Binary?

b) Convert Binary numbers to Grey numbers

11011

11011001101

11001100110

c) Write Java code for the following Algorithm to convert Binary to Grey number:

binary\_to\_grey(n)

if n == 0

grey = 0;

else if last two bits are opposite to each other

grey = 1 + 10 \* binary\_to\_gray(n/10))

else if last two bits are same

grey = 10 \* binary\_to\_gray(n/10))

d) Write Algorithm to convert Grey to Binary Number

e) Write Java code to your Algorithm in (d)

f) Write step-by-step Algorithm to generate n-bit Gray code

g) Apply algorithm to generate 3-bits Gray code

3. An *n*-bit Gray code is a list of the 2*n* different *n*-bit binary numbers such that each entry in the list differs in precisely one bit from its predecessor. The *n* bit binary reflected Gray code is defined recursively. How does algorithm works for n=5, describe step-by-step. Write Java code, compile and run program. <https://introcs.cs.princeton.edu/java/23recursion/>

4. Describe the Array Implementation of Queue with “It was the best of times” example discussed in class. You need to walk through the enqueue and dequeue, and other operations and to manage the front and last pointers. The example shows queue B and queue C, what is the difference, explain.

5. Consider the following QueueOfStrings code to manage queue. The input to this method is

String “Snow storm - - cold today - - - and - - 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();

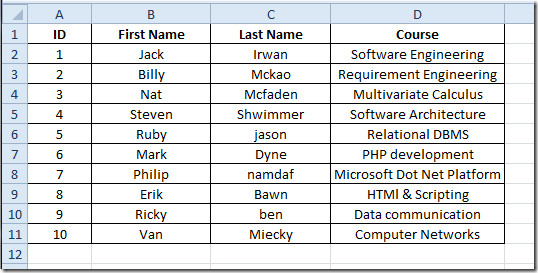
if (s.equals("-")) StdOut.print(q.dequeue());

else q.enqueue(s);

}

}

6. 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: enqueue, dequeue, isEmpty, isFull

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 2 element

—enqueue all elements into queue

—enqueue this element into the queue:

11 John Henry “software development”

12 Raj Manish “Statistician”

13 Justin Morgan “engineering statistics”

—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?

7. Consider signed byte X, and unsigned byte Y. What are the possible values for both X and Y?

8. Java is Pass-by-Value, what does that mean? How does it work with examples,

int X=5; String s=“Testing”; ArrayList = {10, 21, 5, 30, 9, 3}

9. 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

10. Consider this 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) Write Java code to transform this Infix expression to Postfix: (1 + 3 + ( ( 4 / 2 ) \* ( 8 \* 4 ) ))

B) Write Java code to Evaluate Postfix expression.