CSC 212 Midterm 1 - Fall 2013

Exam Duration: 2 Hours

13/11/2013

Question 1 [25 points]

Find the complexity and big-oh for the following two methods (write your answer on the exam answer sheet):

1.

	Statement	S/E	Frequency	Total
1	public static int getNum(int n)			
2	{			
3	n=1;			
4	while(n <5) {			
5	System.out.println("Less than 5");			
6	n++;			
7	}			
8	return n;			
9	}			
	Total operations			
	Big-oh			

2.

	Statement	S/E	Frequency	Total
1	void addThemUp(int n) {			
2	int sum=0;			
3	for(int i=0; i<n; i++) {			
4	for(int j=1; j<=10; j++) {			
5	sum= sum+i+j;			
6	System.out.println(sum);			
7	}			
8	}			
9	System.out.println("Good bye");			
10	}			
	Total operations			
	Big-oh			

. . . . . .

Question 2 [25 points]

1. Complete the implementation of the class *ArrayQueue* below. Write down the methods: *public boolean full(), public int length(), public void enqueue(T e)*, and *public T serve()*.

2. Add to the class the method *public void removeTail()*, that removes the element at the tail, and the method *public void removeHead()* that removes the element at the head (as a precondition to these two methods, the queue must not be empty).

```
public class ArrayQueue<T> {
        private int maxsize;
        private int size;
        private int head, tail;
        private T[] data;

        public ArrayQueue(int n) {
                maxsize = n;
                size= 0;
                head= tail= 0;
                data = (T[]) new Object[maxSize];
        }
        ...
}
```

. . . . . .

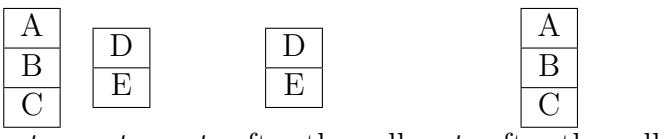
Question 3 [25 points]

1. Write a static method *findSmallest* (user of the ADT queue) that takes as input a non-empty queue q of integers and finds the smallest number in this queue (q must not change). The method signature is: *public static Integer findSmallest(Queue <Integer >q)*. You can compare between variables of type *Integer* using the usual operators: ==, <, >, etc.

   Example 3.1. *If* $q : 2 \to 4 \to 3 \to 8 \to 3 \to 6 \to 1 \to 9$*, then* findSmallest(q) *returns "1", as it is the smallest number in the queue.*

2. Write the static method *exchange* (user of the ADT stack) that takes as input two stacks st_1, st_2 and exchanges their contents. The method signature is: *public static* $<T>$*void exchange (Stack* $<T>st_1$*, Stack*$<T>st_2$*).*

   Example 3.2. *This is an example:*



st_1 st_2 st_1 after the call st_2 after the call

. . . . . .

Question 4 [25 points]

1. Write the static method *firstHalf* (user of the ADT list) that takes a list as input, and returns the first-half of the list (the list must not change). If the number of elements is odd, then the returned half should be the smaller one. The method signature is: *public static* $<T>$*List* $<T>$*firstHalf (List* $<T>$*l)*.

   Example 4.1. *If* $l : A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$*, then* firstHalf(l) *returns:* $A \rightarrow B$.

2. Write the method *removeDuplicates*, member of class *LinkedList*, that removes any duplicates in the list. If an element appears many times, only the first element should be kept, while the duplicates should be removed. Assume that the list is not empty, and use the method *equals* to test for equality. Do not use any auxiliary data structures and do not call any methods. The method signature is *public void removeDuplicates()*.

   Example 4.2. *If the list contains* $A \rightarrow B \rightarrow A \rightarrow A \rightarrow B \rightarrow D \rightarrow E \rightarrow D$*, then after calling* removeDuplicates*, its content becomes* $A \rightarrow B \rightarrow D \rightarrow E$.

. . . . . .

A Specifications

A.1 ADT List

- FindFirst ( ): **requires**: list L is not empty. **input**: none. **results**: first element set as the current element. **output**: none.
- FindNext ( ): **requires**: list L is not empty. Current is not last. **input**: none. **results**: element following the current element is made the current element. **output**: none.
- Retrieve (Type e): **requires**: list L is not empty. **input**: none. **results**: current element is copied into e. **output**: element e.
- Update (Type e): **requires**: list L is not empty. **input**: e. **results**: the element e is copied into the current node. **output**: none.
- Insert (Type e): **requires**: list L is not full. **input**: e. **results**: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. **output**: none.
- Remove ( ): **requires**: list L is not empty. **input**: none. **results**: the current element is removed from the list. If the resulting list is empty current is set to NULL. If successor of the deleted element exists it is made the new current element otherwise first element is made the new current element. **output**: none.
- Full (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L has reached the maximum number allowed then flag is set to true otherwise false. **output**: flag.
- Empty (boolean flag): **requires**: none. **input**: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false. **output**: flag.

- Last (boolean flag): **requires**: L is not empty. **input**: none. **results**: if the last element is the current element then flag is set to true otherwise false. **output**: flag.

A.2 ADT Queue

- Enqueue (Type e): **requires**: Queue Q is not full. **input**: Type e. **results**: Element e is added to the queue at its tail. **output**: none.
- Serve (Type e): **requires**: Queue Q is not empty. **input**: none. **results**: the element at the head of Q is removed and its value assigned to e. **output**: Type e.
- Length (int length): **requires**: none. **input**: none. **results**: The number of elements in the Queue Q is returned. **output**: length.
- Full (boolean flag): **requires**: none. **input**: none. **results**: If Q is full then flag is set to true, otherwise flag is set to false. **output**: flag.

A.3 ADT Stack

- Push (Type e): **requires**: Stack S is not full. **input**: Type e. **results**: Element e is added to the stack as its most recently added elements. **output**: none.
- Pop (Type e): **requires**: Stack S is not empty. **input**: **results**: the most recently arrived element in S is removed and its value assigned to e. **output**: Type e.
- Empty (boolean flag): **input**: none. **results**: If Stack S is empty then flag is true, otherwise false. **output**: flag.
- Full (boolean flag): **requires**: **input**: none. **results**: If S is full then Full is true, otherwise Full is false. **output**: flag.