Computer Science 367 Midterm Exam
Thursday, July 10, 2014

PRINT Last Name: _____ , First: _____

Signature: _____ CS login: _____

This exam is composed of three parts. *Parts A and B must be answered by filling in your choice using a #2 pencil on the separate answer sheet. Part C must be answered in this booklet. If you need extra pages, raise your hand and request them. You must turn in all pages.*

1. ***Print and sign your name*** *above, and fill in the other information. By doing so, you are affirming academic integrity in the course of this exam (i.e. no academic misconduct).*
2. ***Fill in your name and student ID on the separate answer sheet as well****. All information must be complete and legible.*
3. *Check that you have the complete exam booklet (14 pages total)*
4. *Do not open the booklet until instructed.*
5. *You may not use books, notes, calculators, laptops, PDAs, cellphones, or any other electronic devices on this exam.* ***Turn off and put away all electronics now*** *(before the exam begins).*
6. *This exam is intended to take the full class hour (up to 12:15 PM). Make sure to* ***budget your time wisely****; some questions may have equal value but can take longer to answer than others.*
7. *If you need something clarified please bring it to an instructor's attention. We will announce all clarifications and write them on the board.*

Part	# of questions	Format	Maximum Points	Score
A	12	Multiple Choice	12	
B	6	Multiple Choice	18	
C	3	Written Answer	20	
		Total:	50	

Part A Multiple Choice [12 questions, 1 point each, 12 total points]

For questions 1 through 12, choose the one best answer after reading all the choices. Mark the corresponding letter on your answer sheet.

1) When it comes to variable assignment and passing variables as arguments to methods, Java is:

A) Unpredictable
✔ **B) Pass-by-value**
C) Pass-by-reference
D) Pass-by-reference for Objects and pass-by-value for primitives
E) Mostly pass-by-value

2) In order to use a data structure, one needs to know how it is implemented:

A) True
✔ **B) False**

3) Checked exceptions do not need to be declared in the method header:

✔ **A) True**
B) False

4) Which of the following is a better choice for storage in a list implementation where the main use case is the `get(int)` operation (i.e. this is the operation that will be called most frequently):

✔ **A) Array**
B) Linked List
C) Tree
D) Exception
E) Recursion

5) An algorithm with an O(2n) running time has the same complexity as another one with O(n) running time:

✔ **A) True**
B) False

6) What is the worst-case complexity for the `remove(int)` operation on a list that is implemented using an array?

A) O(1)
B) O(logn)
✔ **C) O(n)**
D) O(n ^ 2)

7) Which of the following is NOT a benefit of using generics in Java?

A) They provide compile-time type safety
B) They allow code re-use
✔ **C) They never throw exceptions**
D) They allow for simpler code by removing the need for casting

8) What kind of data structure is most suited in a printer for holding the pages of a document that has been sent for printing?

A) Linked List
B) Array
C) Stack
✔ **D) Queue**
E) Java Object

9) Assume `L1` and `L2` are two `ArrayList` objects, containing `n` and `m` items respectively. Also assume that the underlying arrays have enough unused space so that they never need to be resized for now. `elt` is an object of the same type as those in `L1` and `L2`. What is the worst-case time complexity of the following code fragment:

```
if (L2.contains(elt))
     L1.add(item);
```

A) O(1)
✔ **B) O(m)**
C) O(n)
D) O(m+n)
E) O(m*n)

10) Suppose you implemented a Stack ADT using an array (non-circular) where the top element of the stack is at index 0. What is the worst-case complexity for `pop()`?

A) O(1)
B) O(logn)
✔ **C) O(n)**
D) O(n^2)

11) Suppose `L` is a List that initially contains the following Strings in order: ["S", "U", "M", "M", "E", "R"]. What will the contents of `L` be after the following code fragment is executed on it?

```
for (int i=0; i < L.size(); i++) {
     L.add(i, L.remove(0));
```

A) ["R", "E", "M", "M", "U", "S"]

B) ["S", "U", "M", "M", "E", "R"]

C) ["S", "R", "U", "E", "M", "M"]

✔ **D) ["M", "S", "E", "M", "R", "U"]**

E) ["U", "M", "S", "E", "M", "R"]

12) Recall that "instantiation" means to create an object. In Java, an interface:

✔ **A) Cannot be instantiated but can be the declared type of an object**
B) Cannot be instantiated and cannot be the declared type of an object
C) Can be instantiated and can be the declared type of an object
D) Is a special type of abstract class that can be instantiated

Part B Multiple Choice [6 questions, 3 points each, 18 total points]

For questions 13 through 18, choose the one best answer after reading all the choices. Mark the corresponding letter on your answer sheet.

13) What does the following code fragment do to a queue `q`?

```
Stack<String> s = new Stack<String>();
while(!q.isEmpty())
     s.push(q.dequeue());
while(!s.isEmpty())
     q.enqueue(s.pop());
```

A) Empties the queue
✔ **B) Reverses the items on the queue.**
C) Transfers the items in the queue to the stack in order
D) Nothing. The queue contains exactly the same items as before in the same order.

14) Suppose `L` is a list of strings with the following items in order:
["a", "b", "c", "d", "x", "y", "z", "w"]
and we run the following code snippet on it:

```
Stack<String> s = new Stack<String>();
for (int i = 0; i < L.size(); i++)
     s.push(L.remove(i));

while (!s.isEmpty())
     System.out.print(s.pop());
```

What gets printed to the console?:

A) "abcd"
B) "wzyxdcba"
✔ **C) "zxca"**
D) "bdywacxz"
E) "wzyxdcba"

15) Suppose that an *intermixed* sequence of (stack) push and pop operations are performed on an initially empty stack. The pushes push the integers 0 through 9 in order; the pops print out the return value in addition to removing the topmost element on the stack. Which of the following sequences **cannot** occur?

A) 4 3 2 1 0 9 8 7 6 5

B) 2 5 6 7 4 8 9 3 1 0

C) 4 3 2 1 0 5 6 7 8 9

D) 1 2 3 4 5 6 9 8 7 0

✔ **E) 4 6 8 7 5 3 2 9 0 1**

16) Suppose that an *intermixed* sequence of enqueue and dequeue operations are performed on an initially empty queue. The enqueue operations put the integers 0 through 9 in order onto the queue; the dequeue operations print out the return value in addition to removing the element at the head of the queue. Which of the following sequences is a valid output for this scenario?

A) 4 3 2 1 0 5 6 7 8 9

B) 4 6 8 7 5 3 2 9 0 1

C) 2 5 6 7 4 8 9 3 1 0

✔ **D) 0 1 2 3 4 5 6 7 8 9**

E) 2 4 6 5 7 3 0 9 1 8

17) Consider the following code snippet where j and n are ints and n is a power of 2:

```
j = 1;
while (j <= n)
     j = j*2;
```

How many times is the while condition checked? (All logs are with base 2)

A) n B) logn + 1 C) n/2 – 1 D) logn ✔ **E) logn + 2**

18) Suppose `n` is a reference to a node in a **doubly-linked** list that contains at least three nodes, and that `n` is neither the first nor the last node in the list. If `newNode` is a new node (with null `prev` and `next` pointers), which of the following code fragments correctly inserts it immediately *before* `n`?

A)
```
n.getPrev().setNext(newNode);
n.setPrev(newNode);
newNode.setNext(n);
newNode.setPrev(n.getPrev());
```

B)
```
n.setPrev(newNode);
n.getPrev().setNext(newNode);
newNode.setNext(n);
newNode.setPrev(n.getPrev());
```

✔ **C)**
```
newNode.setNext(n);
newNode.setPrev(n.getPrev());
n.setPrev(newNode);
newNode.getPrev().setNext(newNode);
```

D)
```
newNode.setNext(n);
newNode.setPrev(n.getPrev());
n.setPrev(newNode);
n.getPrev().setNext(newNode);
```

E)
```
n.setPrev(newNode);
newNode.getPrev().setNext(newNode);
newNode.setNext(n);
newNode.setPrev(n.getPrev());
```

Part C Written Answers [3 questions, 20 total points]

Write your answers to questions 19 through 21 in this booklet. If you need more space, use the back of the page and indicate that your work continues on the back.

19) [6 points] Assume that the recursion depth limit on a computer depends only on the *number* of activation records on the call stack (i.e. not on how big each one is). Write a Java method `findStackDepth` that *gracefully* finds the maximum recursion limit on a computer. Your method must return an `int`. *Hint: Use recursion and exceptions. One exception you may be interested is called* `StackOverflowError`.

```
private static int findStackDepth() {
    try {
        return 1 + findStackDepth();
    } catch (StackOverflowError e) {
        return 0;
    }
}
```

20) [8 points] Design a queue-like data structure that supports the following operations:

```
void enqueue(E)
E dequeue()
boolean isEmpty()
E removeKth()
```

The first three operations behave exactly the way they would in a queue. The last operation `removeKth` is a special operation that returns the kth element from the head of the queue, **and** in addition dequeues it and everything ahead of it in the queue. If the queue contains less than k elements, it throws an exception.

Describe how you would implement this data structure and the complexities of each operation. You will be awarded points based on correctness and low time complexity. Be sure to describe the storage mechanism, accounting variables (i.e. class members), how they are maintained with each operation, boundary cases, etc. You can describe them at a fairly high-level as long as you mention the critical details required. *You are not required to write code for this problem.*

21) [6 points] You are tasked as an implementer to add a `find` method to a linked list implementation that searches for a given object and returns its position in the linked list. If the element is not found or the list is empty, it returns -1. The linked list class only has the following data member:

```
private ListNode<E> head;
```

Assume that the implementation **maintains a dummy header**. You may assume that the list does not contain `null` *items,* and that items contained in the list can be compared using the `equals()` method. Complete the following partial implementation: (you must fill in all the blanks and the body of the while loop)

```
public int find(E item) {

     // first boundary case

     if ( head.getNext() == null ) return -1;

     // initialize curr pointer and position
     ListNode<E> curr = head;

     int position = -1;


     while ( curr.getNext() != null ) {    // while not at the
                                            // end of the list
          curr = curr.getNext();
          position++;
          if (curr.getData().equals(item)) return position;

     }

     // the element was not found if we've gotten here
     return -1;
}
```