

CSCI 136 Written Exam #0
Fundamentals of Computer Science II
Spring 2012

Name: _____

This exam consists of 6 problems on the following 8 pages.

You may use your single-side hand-written 8 ½ x 11 note sheet during the exam. You may use a simple handheld calculator. No computers, mobile devices, cell phones, or other communication devices of any kind are permitted.

If you have a question, raise your hand and I will stop by. Since partial credit is possible, **please write legibly and show your work.**

Problem	Points	Score
1	8	
2	14	
3	8	
4	8	
5	16	
6	12	
Total	66	

1. **Loops, Input** (8 points). Consider the following program:

```
public class Prob1
{
    public static void main(String [] args)
    {
        int i = Integer.parseInt(args[0]);
        while (i > 0)
        {
            System.out.print(i + ",");
            i = i - 2;
        }
        System.out.println(i);
    }
}
```

Below are four example executions of the program. Give the output produced by the program. If the given input would cause a runtime error, write "runtime error". If it would cause a compile error, write "compiler error".

Command line	Output
% java Prob1 10	10,8,6,4,2,0
% java Prob1 3 foo 3.14	3,1,-1
% java Prob1 -10	-10
% java Prob1 -10.0	Runtime error

2. **ADTs and Data Structures** (14 points).

a) For each problem on left, give the letter of the **best** abstract data type for solving that problem. Each letter will be used **exactly once**.

Problem	Best letter	ADT
Implement the <i>Back</i> button on a web browser.	a	a) Stack b) Queue c) List d) Map e) Set
Given a unique username, look up the last login time for that user.	d	
Given a passport number, determine if that number is in a known group of terrorists (some agency has given you a list of passport numbers of terrorists).	e	
Simulate the behavior of cars at a traffic light	b	
An online movie site stores many details about each title it has available. Among the details is a rating from 1 to 5 stars. The web interface needs to sort movies according to their rating.	c	

b) Your boss asks you to implement a queue ADT that stores doubles. Give an advantage and a disadvantage for each of the following possible underlying data structures:

Data structure	Advantage	Disadvantage
Fixed array	Simple to implement. Uses less space if you know the exact size you need.	Has a fixed maximum size, wastes memory if you don't have this much data.
Doubling array	You don't have to worry about hitting some maximum size.	In the worst case, wastes half the memory. Occasionally, has to do an expensive doubling or shrinking of the array.
Linked list	You don't have to worry about hitting some maximum size. Smart implementation can always dequeue and enqueue in $O(1)$ time.	More complicated to implement. Requires memory for each "next" reference link.

c) Your boss is troubled by your "complicated" class declaration:

```
public class MyQueue<E>
```

Explain to your boss the advantage of declaring your class in the above way rather than like this:

```
public class MyQueue
```

By declaring your class using Java generics, you allow MyQueue to store any reference type and only need to write and debug a single class for all your queue-related needs.

3. **Performance** (8 points). The following table gives running times measured for a program using an input size of N, for various values of N.

N	time (seconds)
1000	0.05
2000	0.17
4000	0.66
8000	2.65
16000	10.52

a) Which of the following best describes the order of growth of the running time of this program? **Circle one** of the following:

- I. $O(1)$, constant
- II. $O(\log N)$, logarithmic
- III. $O(N \log N)$, linearithmic
- IV. $O(N)$, linear
- V. $O(N^2)$, quadratic $10.52 / 2.65 = 3.96981132 \approx 4 = 2^2$
- VI. $O(N^3)$, cubic
- VII. $O(2^N)$, exponential

b) Give the equation showing the running time of the program in seconds as a function of the input size N (you need to find the leading constant).

$$T = a N^2$$

$$10.52 = a (16000)^2$$

$$a = 4.10938 \times 10^{-8}$$

$$T = 4.10938 \times 10^{-8} N^2$$

c) Estimate the program's running time in seconds for an input size of $N = 100,000$.

$$T = 4.10938 \times 10^{-8} N^2$$

$$T = 4.10938 \times 10^{-8} (100000)^2$$

$$T = 410.9375 \text{ seconds}$$

4. **Performance** (8 points). For each code section in the left column, circle the letter in the right-column that **best** describes the order of growth of the code. **Circle one letter**. You have no idea what goes on in the `mystery` method. Do not worry about what is being calculated in the variable `num` or whether it overflows/underflows.

<pre>for (int i = 0; i < N; i++) { for (int j = 0; j < N; j++) { for (int k = 0; k < 4200; k++) { num = num / 2 + 5; ++num; } } }</pre>	<ul style="list-style-type: none"> a) O(1) b) O(N) c) O(N²) d) O(N³) e) O(N) or worse (slower) f) O(N²) or worse (slower) g) O(N³) or worse (slower)
--	--

<pre>for (int i = 0; i < N; i++) { num = num + (i * N); } for (int i = 0; i < N * 10; i++) { num = num % i; }</pre>	<ul style="list-style-type: none"> a) O(1) b) O(N) c) O(N²) d) O(N³) e) O(N) or worse (slower) f) O(N²) or worse (slower) g) O(N³) or worse (slower)
---	--

<pre>for (int i = 0; i < (N / 100); i++) { if (mystery(N, i) < 0) num += i; else num -= N; }</pre>	<ul style="list-style-type: none"> a) O(1) b) O(N) c) O(N²) d) O(N³) e) O(N) or worse (slower) f) O(N²) or worse (slower) g) O(N³) or worse (slower)
--	--

<pre>int i = N / 2; int j = 0; int k = 0; while (i < N) { while (j < N) { while (k <= N) { num += Math.random(); k++; } j++; } i++; }</pre>	<ul style="list-style-type: none"> a) O(1) b) O(N) c) O(N²) d) O(N³) e) O(N) or worse (slower) f) O(N²) or worse (slower) g) O(N³) or worse (slower)
--	--

5. **Linked Structures** (16 points). The class `QueueOfInts` implements a FIFO queue that holds primitive `int` values. The underlying data structure is a `null` terminated linked list. Here is part of the class:

```
public class QueueOfInts
{
    private Node first = null;
    private class Node
    {
        private int item;
        private Node next;
        Node(int item, Node next)
        {
            this.item = item;
            this.next = next;
        }
    }
    public void enqueue(int val)
    {
        if (first == null)
        {
            first = new Node(val, null);
            return;
        }
        Node current = first;
        while (current.next != null)
            current = current.next;
        current.next = new Node(val, null);
    }
    /* more methods go here */
}
```

a) Draw a diagram representing the state of the linked list after each line of the following five line program. Use the block and arrow notation shown in class with a `null` being a dot. Be sure to show where the instance variable `first` is pointing and the value at each node in the list.

Line	Code	Diagram
1	<code>QueueOfInts q = new QueueOfInts();</code>	
2	<code>q.enqueue(2);</code>	
3	<code>q.enqueue(4);</code>	
4	<code>q.enqueue(3);</code>	
5	<code>q.enqueue(-7);</code>	

5. Linked Structures (continued)

b) **Place an X in the one box** containing the `QueueOfInts` `dequeue` method that correctly implements a First-In First-Out (FIFO) queue.

```
public int dequeue()
{
    if (first == null)
        throw new RuntimeException("Empty!");
    int result = first.item;
    first = first.next;
    return result;
}
```

X

```
public int dequeue()
{
    if (first == null)
        throw new RuntimeException("Empty!");
    Node current = first;
    while (current.next.next != null)
        current = current.next;
    int result = current.next.item;
    current.next = null;
    return result;
}
```

```
public int dequeue()
{
    if (first == null)
        throw new RuntimeException("Empty!");
    if (first.next == null)
    {
        int result = first.item;
        first = null;
        return result;
    }
    Node current = first;
    while (current.next.next != null)
        current = current.next;
    int result = current.next.item;
    current.next = null;
    return result;
}
```

```
public int dequeue()
{
    if (first == null)
        throw new RuntimeException("Empty!");
    first = first.next;
    return first.item;
}
```

5. Linked Structures (continued).

The `QueueOfInts` class has the following `insert` method :

```
public void insert(int val)
{
    if ((first == null) || (first.item > val))
    {
        first = new Node(val, first);
        return;
    }
    Node current = first;
    while ((current.next != null) && (current.next.item < val))
        current = current.next;
    current.next = new Node(val, current.next);
}
```

c) Draw a diagram representing the state of the linked list after each line of the following five line program. Use the block and arrow notation shown in class with a `null` being a dot. Be sure to show where the instance variable `first` is pointing and the value at each node in the list.

Line	Code	Diagram
1	<code>QueueOfInts q = new QueueOfInts();</code>	
2	<code>q.insert(2);</code>	
3	<code>q.insert(4);</code>	
4	<code>q.insert(3);</code>	
5	<code>q.insert(-7);</code>	

d) In 2⁴ words or less, describe the goal of the `insert` method.

Inserts integer keeping the queue in ascending order.

6. **Arrays and Collections** (12 points). For each of the following, give the code to declare the necessary variable and give code that adds a single non-null entry into the collection. The first one has been done for you. You can assume the `Animal` class has a default constructor that creates a random animal.

Description	Java variable declaration	Example of adding one non-null item
A fixed-sized list that can hold up to 100 integer values.	<pre>int [] d = new int[100];</pre>	<pre>d[0] = 42;</pre>
A dynamically-sized list that can hold any number of integer values.	<pre>ArrayList<Integer> nums = new ArrayList<Integer>();</pre>	<pre>nums.add(42);</pre>
A fixed-sized list that can hold up to 100 objects of type <code>Animal</code> .	<pre>Animal [] animals = new Animal[100];</pre>	<pre>animals[0] = new Animal();</pre>
A map that can track how many times a particular <code>Animal</code> object has been seen.	<pre>HashMap<Animal, Integer> map = new HashMap<Animal, Integer>();</pre>	<pre>map.put(new Animal(), 42);</pre>
A set that can decide if a given word is in a large list of words. Words are represented by <code>String</code> objects.	<pre>HashSet<Animal> set = new HashSet<Animal>();</pre>	<pre>set.add(new Animal());</pre>