CSCI 136 Written Exam #0 **Name: _____**
Fundamentals of Computer Science II
Spring 2012

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`	
`% java Prob1 3 foo 3.14`	
`% java Prob1 -10`	
`% java Prob1 -10.0`	

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 *Back* button on a web browser.		a) Stack b) Queue c) List d) Map e) Set
Given a unique username, look up the last login time for that user.		
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).		
Simulate the behavior of cars at a traffic light		
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.		

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		
Doubling array		
Linked list		

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
```

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
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).

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

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.

`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;`
`  }`
`  }`
`}`
	a) O(1)
	b) O(N)
	c) $O(N^2)$
	d) $O(N^3)$
	e) O(N) or worse (slower)
	f) $O(N^2)$ or worse (slower)
	g) $O(N^3)$ or worse (slower)

`for (int i = 0; i < N; i++)`
`{`
`  num = num + (i * N);`
`}`
`for (int i = 0; i < N * 10; i++)`
`{`
`  num = num % i;`
`}`
	a) O(1)
	b) O(N)
	c) $O(N^2)$
	d) $O(N^3)$
	e) O(N) or worse (slower)
	f) $O(N^2)$ or worse (slower)
	g) $O(N^3)$ or worse (slower)

`for (int i = 0; i < (N / 100); i++)`
`{`
`  if (mystery(N, i) < 0)`
`  num += i;`
`  else`
`  num -= N;`
`}`
	a) O(1)
	b) O(N)
	c) $O(N^2)$
	d) $O(N^3)$
	e) O(N) or worse (slower)
	f) $O(N^2)$ or worse (slower)
	g) $O(N^3)$ or worse (slower)

`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++;`
`}`
	a) O(1)
	b) O(N)
	c) $O(N^2)$
	d) $O(N^3)$
	e) O(N) or worse (slower)
	f) $O(N^2)$ or worse (slower)
	g) $O(N^3)$ 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	`QueueOfInts q = new QueueOfInts();`	
2	`q.enqueue(2);`	
3	`q.enqueue(4);`	
4	`q.enqueue(3);`	
5	`q.enqueue(-7);`	

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;
}
```

```
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	`QueueOfInts q = new QueueOfInts();`	
2	`q.insert(2);`	
3	`q.insert(4);`	
4	`q.insert(3);`	
5	`q.insert(-7);`	

d) In 2^4 words or less, describe the goal of the `insert` method.

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.	`int [] d = new int[100];`	`d[0] = 42;`
A dynamically-sized list that can hold any number of integer values.		
A fixed-sized list that can hold up to 100 objects of type `Animal`.		
A map that can track how many times a particular `Animal` object has been seen.		
A set that can decide if a given word is in a large list of words. Words are represented by `String` objects.		