CIS 110 Spring 2014 — Introduction to Computer Programming
12 May 2014 — Final Exam

Name: _____

Recitation # (e.g., 201): _____

Pennkey (e.g., eeaton): _____

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

_____ _____
Signature Date

Instructions:

- **Do not open this exam until told by the proctor.** You will have exactly 120 minutes to finish it.
- **Make sure your phone is turned OFF (not to vibrate!) before the exam starts.**
- Food, gum, and drink are strictly forbidden.
- **You may not use your phone or open your bag for any reason**, including to retrieve or put away pens or pencils, **until you have left the exam room**.
- This exam is closed-book, closed-notes, and closed-computational devices.
- If you get stuck on a problem, it may be to your benefit to move on to another question and come back later.
- All code must be written out in proper Java format, including all curly braces and semicolons.
- Do not separate the pages. If a page becomes loose, reattach it with the provided staplers.
- Staple all scratch paper to your exam. Do not take any sheets of paper with you.
- If you require extra paper, please use the backs of the exam pages or the extra pages provided at the end of the exam. **Clearly indicate on the question page where the graders can find the remainder of your work (e.g., "back of page" or "on extra sheet")**.
- Use a pencil, or blue or black pen to complete the exam.
- If you have any questions, raise your hand and a proctor will come to answer them.
- When you turn in your exam, you may be required to show ID. **If you forgot to bring your ID, talk to an exam proctor immediately**.
- We wish you the best of luck. Have a great summer!

Scores: [For instructor use only]

Question 0		1 pts
Question 1		10 pts
Question 2		8 pts
Question 3		12 pts
Question 4		9 pts
Question 5		15 pts
Question 6		11 pts
Question 7		10 pts
Question 8		8 pts
Total:		84 pts

TOY Reference Card

INSTRUCTION FORMATS

```
            | . . . . | . . . . | . . . . | . . . . |
  Format 1: | opcode  |    d    |    s    |    t    |  (0-6, A-B)
  Format 2: | opcode  |    d    |       addr        |  (7-9, C-F)
```

ARITHMETIC and LOGICAL operations

```
    1: add              R[d] <- R[s] +  R[t]
    2: subtract         R[d] <- R[s] -  R[t]
    3: and              R[d] <- R[s] &  R[t]
    4: xor              R[d] <- R[s] ^  R[t]
    5: shift left       R[d] <- R[s] << R[t]
    6: shift right      R[d] <- R[s] >> R[t]
```

TRANSFER between registers and memory

```
    7: load address     R[d] <- addr
    8: load             R[d] <- mem[addr]
    9: store            mem[addr] <- R[d]
    A: load indirect    R[d] <- mem[R[t]]
    B: store indirect   mem[R[t]] <- R[d]
```

CONTROL

```
    0: halt             halt
    C: branch zero      if (R[d] == 0) pc <- addr
    D: branch positive  if (R[d] >  0) pc <- addr
    E: jump register    pc <- R[d]
    F: jump and link    R[d] <- pc; pc <- addr
```

Register 0 always reads 0.
Loads from mem[FF] come from stdin.
Stores to mem[FF] go to stdout.

0.) THE EASY ONE (1 point total)

- Check to make certain that your exam has all 13 pages (excluding the cover sheet).
- Write your name, recitation number, and PennKey (username) on the front of the exam.
- Sign the certification that you comply with the Penn Academic Integrity Code.

1.) MULTIPLE CHOICE (10 points total)

1.1) (4 points) For each scenario below, choose whether it would be better to store the data in an array, an `ArrayList`, or a `LinkedList`. Circle one answer for each scenario.

(a) The data rarely changes, and the program should use the least amount of memory possible:
`array` `ArrayList` `LinkedList`

(b) The amount of data will vary over time, but memory usage should **always** be minimal:
`array` `ArrayList` `LinkedList`

(c) The order of the data is constantly changing:
`array` `ArrayList` `LinkedList`

(d) The data is PennIDs of students, updated daily, and repeatedly searched using binary search:
`array` `ArrayList` `LinkedList`

1.2) (2 points) Which algorithm for sorting a large array of integers is the fastest?

(a) Insertion sort

(b) Merge sort

(c) Selection sort

(d) Bubble sort

(e) They are all equally fast

1.3) (2 points) Which of the following expressions is true? (Circle only one)
(Recall the following bitwise operators: `&` (and), `|` (or), `^` (exclusive or).)

(a) `(1 ^ 0) & 1 == 1`

(b) `(0 & 1) == 1`

(c) `1 ^ (0 ^ 1) == 1`

(d) `!((0 | 1) == 1)`

(e) `(1 | (1 ^ (1 & 0))) != 1`

1.4) (2 points) In TOY, the program counter represents:

(a) The number of available memory slots

(b) The number of lines of code in the program

(c) The amount of time the program has been running

(d) The memory address of the instruction that is currently being executed

(e) The number of CPU clock tics since the computer last rebooted

2.) METHOD OVERLOADING (8 points total)

Your start-up's backers have hired a seasoned executive to run you company. He has proposed the following API for the `add()` methods for a `LinkedList` of integers, but he has forgotten to consider Java's rules for method overloading:

(A) `public void add(int x)`
(B) `public void add(int index, int x)`
(C) `public void add(int x, int index)`
(D) `public boolean add(int x, int index)`
(E) `public void add(int x, boolean duplicates)`
(F) `private boolean add(int x)`

2.1) (5 points) Circle `Yes` or `No` to indicate whether each of the pairs of methods listed below could both be included in the same Java API:

(A) and (B): `Yes` `No`

(A) and (F): `Yes` `No`

(B) and (C): `Yes` `No`

(C) and (D): `Yes` `No`

(D) and (E): `Yes` `No`

2.2) (3 points) After some debate, you decide to ignore your seasoned executive, and you condense the API to the following:

```
public LinkedList()                      public int get(int index)
public void add(int x)                   public int size()
public void add(int x, int index)        public void remove(int index)
```

One hallmark of good programming is building upon other methods that you've already written. Provide an implementation of `add(int x)` that relies solely on the other methods from the condensed API above. Do not access any fields of the linked list directly.

```
public void add(int x) {




}
```

3.) LISTS AND METHODS AND COMPLEXITY, O(MY)! (12 points total)

For each list operation, circle the computational complexity for each data structure. Make certain that you have circled one complexity in both columns within each row. The first row has been completed for you. For `ArrayLists`, you may assume that the underlying array does not need to be resized. You should also assume efficient implementations for all methods and data structures.

	`ArrayList`		`DoublyLinkedList`	
Get the element at the head of the list	$\boxed{O(1)}$	$O(n)$	$\boxed{O(1)}$	$O(n)$
Construct an empty list	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Get the size of the list	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Add an element to the head of the list	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Add an element to the tail of the list	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Get the element at index i in the list	$O(1)$	$O(n)$	$O(1)$	$O(n)$
Remove the element at index `0`	$O(1)$	$O(n)$	$O(1)$	$O(n)$

4.) `REMOVE(REMOVE(REMOVE(REMOVE(...))))` (9 points total)

Fill in the blanks in the partial recursive `LinkedList` implementation below to complete the class. Each of your answers should be one line of code or less. You may assume that all other methods of the class have already been completed.

```
public RecursiveLinkedListOfStrings {
    private Node head;
    private int size;

    private class Node {
        public String value;
        public Node next;
        public Node(String v, Node n) {  value = v;  next = n;  }
    }

    ...   // other LinkedList methods would go here

    /* Removes the first instance of the specified value from the list */
    public void remove(String value) {
        ________________________________________________ ;   /* LINE A */
    }

    /* Removes the first node containing s from the sublist rooted at n.
     * n may be null if we are at the end of the list.
     * Returns the sublist, with the specified value removed
     */
    private Node recursiveRemove(Node n, String s) {
        if (n == null) {                // reached the end of the list
            return ______________________________________ ;   /* LINE B */
        } else if (n.value.equals(s)) {  // found a matching node
            size--;
            return ______________________________________ ;   /* LINE C */
        }
        n.next = recursiveRemove(n.next, s);
        return n;
     }
}
```

Write your answers in the spaces below:

(a) Line A: _____

(b) Line B: _____

(c) Line C: _____

5.) FUN-SIZE STACK PACK (15 points total)

Write a `Stack` class that conforms to the following API:

```
public class Stack
-------------------------------------------------------------------------
public Stack()              // create an empty stack
public void push(String s)  // push a new element on the stack
public String pop()         // remove and return the top element
                            //   return null if the stack is empty
public String top()         // peek at the top element without removing it
                            //   return null if empty
public boolean isEmpty()    // return whether the stack is empty
```

Make certain that all of your methods take $O(1)$ time. You do not need to comment your code, but you may if you wish to clarify your implementation.

You must write your `Stack` class **without** defining a separate *Node* class. Instead you must build it using an existing `LinkedList` class. In other words, your `Stack` object will contain a `LinkedList` object as a field, and will store all data in this `LinkedList`. Your `Stack` class must rely on the `LinkedList`'s methods, and must not create or manipulate individual nodes directly.

You should assume that you have a fully working `LinkedList` class that conforms to the API below. If `index` is ever invalid (less than 0 or greater than `size()`), the `LinkedList` methods will throw an `InvalidArgumentException`.

```
public class LinkedList
-----------------------------------------------------------------------------
public LinkedList()                   // create an empty list
public void add(String s)             // add s to the end of the list
public void add(String s, int index)  // add s to the list at the specified index
public String get(int index)          // get the element at the specified index
public String remove(int index)       // remove the element at the specified index
public int size()                     // get the total number of elements
```

WRITE YOUR ANSWER ON THE NEXT PAGE

Fun-Size Stack Pack (Cont'd)

Write your `Stack` class on this page. If you have work on this page that is not part of your solution, make sure to **indicate your final answer very clearly**. If it is not very clear which code is your final answer (including if you give two different version), we will grade the code that is closest to the top of the page.

6.) ODE TO TOY (11 points total)

Your friendly TAs wrote the following TOY program for fun during our end-of-semester staff picnic. Unfortunately, they did a lousy job. They didn't write proper comments or document what the program does, and there seems to be a serious bug that makes program behave unpredictably. At least the auto-generated assembly comments are correct.

```
10: 8AFF   read R[A]
11: 8BFF   read R[B]
12: 21AB   R[1] <- R[A] - R[B]
13: 9BFF   write R[B]
14: C121   if (R[1] == 0) goto 21
15: 7373   R[3] <- 0073
16: 4333   R[3] <- R[3] ^ R[3]
17: 93FF   write R[3]
18: 1AAA   R[A] <- R[A] + R[A]
19: 211B   R[1] <- R[1] - R[B]
20: C014   if (R[0] == 0) goto 14
21: 8CFF   read R[C]
22: 3CBC   R[C] <- R[B] & R[C]
23: 9CFF   write R[C]
24: 0000   halt
```

6.1) (2 points) In **20 words or less**, describe the bug that makes program behave unpredictably.

6.2) (3 points) Correct the bug you described above. Write out the corrected line of code for each line that changes. You do not need to write out the lines that do not change, and you do not need to write out the auto-generated assembly comments (or any other comments).

CONTINUES ON NEXT PAGE

6.3) (6 points) Given the following input, print the output you would receive once the bug has been fixed. If you believe the original program listing is correct, give the output you think it will produce. You may write your answers in base 10 (decimal) or base 16 (hexadecimal).

(a) `1 1 8`

(b) `6 2 8`

(c) `5 2 9`

7.) A RAY IF I DO, A RAY IF I DON'T (10 points total)

The following function takes a string and a length as arguments, and returns an array of the specified length containing the string's characters.

```
1 public static char[] arrayify(String str, int length) {
2     char[] chars = new char[length];
3     for (int i = 0; i < str.length(); i++)
4         chars[i] = str.charAt(i);
5     return chars;
6 }
```

For example, `arrayify("110 is a ray of sun.", 20)` would return the 20-element array:

```
{ '1', '1', '0', ' ', 'i', 's', ' ', 'a', ' ', 'r', 'a', 'y', ' ', 'o', 'f', ' ', 's', 'u', 'n', '.' }
```

7.1) (5 points) For each of the calls to `arrayify` in the left column, write the <u>one</u> letter from the right column of the exception it would cause (or indicating there would be none):

_____ `arrayify("hi", 2);`

_____ `arrayify("hi", 1);`

_____ `arrayify("hi", 5);`

_____ `arrayify(null, 0);`

_____ `arrayify(null, 1);`

(a) `java.lang.ArrayIndexOutOfBoundsException`

(b) `java.lang.NullPointerException`

(c) `java.lang.IllegalArgumentException`

(d) This statement will not cause an exception.

7.2) (5 points) Write a version of `arrayify()` that returns `null` instead of throwing exceptions. The return value should match the version of `arrayify()` above, except when an exception would have been thrown. If your implementation reuses lines of code that already appear above, write "**Line X**" instead of rewriting that line of code. For instance, write "**Line 5**" instead of "`return chars;`"

```
public static char[] arrayify(String str, int length) {
```

```
}
```

8.) BAIT AND SWITCH (8 points total)

The following code models fishers who fish for fish in the sea. They aren't very good though, and sometimes they hook each other instead. This code is completely correct (and tested), except for one line in the `Fish` class. Sadly, the fishers aren't much better at programming than they are at fishing.

```
public interface Catchable {
    public void rename(String name);
}

public class Fish implements Catchable {
    private String  name;

    public  Fish(String name)        { rename(name); }
    public  void rename(String name) { name = name;  }
    public  String toString()        { return name;  }
}

public class Fisher implements Catchable {
    private String  name;

    public  Fisher(String name)      { this.name = name; }
    public  void rename(String name) { /* do nothing */  }
    public  String toString()        { return name;      }

    public Catchable fish(Catchable[] fish) {
        int randomFish = (int) (fish.length * Math.random());
        if (randomFish < fish.length - 1) {
            System.out.println(name + " caught " + fish[randomFish]);
            return fish[randomFish];
        } else {
            System.out.println(name + " caught nothing");
            return null;
        }
    }
}
```

8.1) (3 points) Find and correct the single line of code in the `Fish` class that behaves incorrectly. You only need to write the corrected line of code below; you do not need to write any of the surrounding code or explain the error.

CONTINUES ON THE NEXT PAGE

Bait and Switch (Cont'd)

8.2) (5 points) The following program models a "catch-and-release" fishing expedition. Anything that is caught is immediately released and can be caught again. Fill in the blanks below in the program's output. Assume the random values that are assigned to the variable `randomFish` are, in order: **3, 2, 3, 4, 1, 2, 3, 4, 4, 1**.

```
public class BaitAndSwitch {
    public static void main(String[] args) {
        Fisher carrie = new Fisher("Carrie");
        Fisher rod    = new Fisher("Rod");

        Fish wanda   = new Fish("Wanda");
        Fish stanley = new Fish("Stanley");
        Fish herring = new Fish("This is a red herring");

        Catchable[] sea       = { carrie, rod, wanda, stanley, herring };
        String[]    fishNames = { "Bernie", "Luke", "Leia", "Han", "Debbie" };

        for (int i = 0; i < 5; i++) {
            Catchable f = carrie.fish(sea);
            if (f != null) // Carrie likes to name her fish
                f.rename(fishNames[i]);
            rod.fish(sea);
        }
    }
}
```

_____ caught _____

_____ caught _____

_____ caught _____

_____ caught _____

_____ caught _____

_____ caught _____

_____ caught _____

_____ caught _____

_____ caught _____

_____ caught _____

Have a great summer!

[**Scrap Paper: This Page Intentionally Blank**]