

UC Berkeley – Computer Science
CS61B: Data Structures

Midterm #1, Spring 2017

This test has 10 questions worth a total of 80 points, and is to be completed in 110 minutes. The exam is closed book, except that you are allowed to use one double sided written cheat sheet (front and back). No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **Write the statement out below in the blank provided and sign. You may do this before the exam begins.**

“I have neither given nor received any assistance in the taking of this exam.”

Signature: _____

#	Points	#	Points
0	0.5	5	8
1	4.5	6	12
2	7	7	0
3	6	8	8
4	12	9	8
		10	14
		TOTAL	80

Name: _____Chalisee Fahwajiarns_____

SID: _____6054756961_____

Three-letter Login ID: _____why_____

Login of Person to Left: _____soo_____

Login of Person to Right: _____far_____

Exam Room: _____A1 Hearst Annex_____

Tips:

- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- There are a lot of problems on this exam. **Work through the ones with which you are comfortable first. Do not get overly captivated by interesting design issues or complex corner cases you’re not sure about.**
- Not all information provided in a problem may be useful.
- Unless otherwise stated, all given code on this exam should compile. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we’ll announce a fix. Unless we specifically give you the option, the correct answer is not ‘does not compile.’
- indicates that only one circle should be filled in.
- indicates that more than one box may be filled in.
- For answers which involve filling in a or , please fill in the shape completely.
- When the exam says “write only one statement per line”, a for loop counts as one statement.

Optional. Mark along the line to show your feelings
 on the spectrum between ☹ and ☺.

Before exam: [☹_____☺].
 After exam: [☹_____☺].

0. So it begins (0.5 points). Write your name and ID on the front page. Write the exam room. Write the IDs of your neighbors. Write the given statement. Sign when you're done with the exam. Write your login in the corner of every page. Enjoy your free half point ☺.

1. Pass By Value (4.5 points). Assume we run the `main` method. In the blanks next to each print statement, write the result of the statement. If the print statement executes multiple times, use multiple blanks. For example, if `swap(int x, int y)` gets used 3 times, use the first blank to list what it prints the first time, the second blank to say what it prints the second time, etc. **You may not need all the blanks.**

```
public static void swap(int x, int y) {
    int temp = y;
    y = x;
    x = temp;
    System.out.println(x);      12           4           _____
}
public static void swap(int[] x, int[] y) {
    int[] temp = y;
    y = x;
    x = temp;
    System.out.println(x[0]);   4           _____           _____
}
public static void main(String[] args) {
    int x1 = 42, y1 = 12;
    swap(x1, y1);
    System.out.println(x1);     42
    System.out.println(y1);     12
    int[] x2 = new int[] {1, 2, 3};
    int[] y2 = new int[] {4, 5, 6};
    swap(x2, y2);
    System.out.println(x2[0]);  1
    System.out.println(y2[0]);  4
    x2 = new int[] {1, 2, 3};
    y2 = new int[] {4, 5, 6};
    swap(x2[0], y2[0]);
    System.out.println(x2[0]);  1
    System.out.println(y2[0]);  4
}
```

Login: _____

2. The Return of Pass By Value (7 points). Fill in the print statement blanks below.

```

public class IntList5000 {
    private IntList5000 rest;
    private int primitive;
    private Integer reference;
    public static String slogan;

    public IntList5000(IntList5000 r, int p, Integer r, String s) {
        rest = r;    primitive = p;    reference = r;    slogan = s;
    }

    public static void disenturbulate(IntList5000 p) {
        p.primitive = 5000;
        p.reference = new Integer(1000);
        p = new IntList5000(null, 5, 10, "relax");
    }

    public void indoctrinate(int b) {
        b = 0;
        this.rest.primitive = b;
    }

    public void integrate(Integer v) {
        this.reference = v;
        v = new Integer(10);
    }

    public static void main(String[] args) {
        IntList5000 x = new IntList5000(null, 100, new Integer(7), "live");
        IntList5000 y = new IntList5000(x, 9, x.reference, "eat");
        System.out.println(y.primitive);           9
        System.out.println(y.reference);           7
        IntList5000.disenturbulate(y);
        System.out.println(y.primitive);           5000
        System.out.println(y.reference);           1000
        y.indoctrinate(100);
        System.out.println(x.primitive);           0
        x.integrate(new Integer(200));
        System.out.println(x.reference);           200
        System.out.println(IntList5000.slogan);    relax
    }
}

```

3. Potpourri (6 Points).

a) Suppose `Cat` and `Dog` are subclasses of `Animal`. All three classes have default (zero-argument) constructors. For each answer below, mark whether it causes a compilation error, runtime error, or runs successfully. Consider each line independently of all other lines. **Fill in bubbles completely.**

Compilation Error	Runtime Error	Runs Fine	
<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<code>Cat c = new Animal();</code>
<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<code>Animal a = new Cat();</code>
<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<code>Dog d = new Cat();</code>
<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<code>Animal a = (Cat) new Cat();</code>
<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<code>Dog d = (Dog) new Animal();</code>


b) For each of the following, if it is an error that causes the Java **compiler** `javac` to abort compilation, fill in the square. **Fill in squares completely.**

- Missing semicolon at end of statement.
- Division by an `int` variable whose value may be zero.
- Attempting to instantiate a generic array as `new T[10]`, where `T` is a type parameter.
- Attempting to access `x[i]` when array `x` has length 3 and `i` is 10.
- Missing return type for a static method.

c) Consider the code below:

```
public static void f() {
    System.out.println(2);
    System.out.println(3);
}

public static void main(String[] args) {
    System.out.println(1);
    f(); // BREAKPOINT
    System.out.println(4);
}
```

IntelliJ expert Omid places a breakpoint on the line marked "BREAKPOINT", then runs the main method above in "Debug" mode. Upon reaching the breakpoint, Omid clicks "Step Over"  exactly once). Write the output of the program (including text printed before the breakpoint), assuming Omid does nothing else after clicking "Step Over".

1

2

3

Login: _____

4. Flirbocon (12 points). Consider the declarations below. Assume that Falcon extends Bird.

```
Bird bird = new Falcon();
Falcon falcon = (Falcon) bird;
```

Consider the following possible features for the Bird and Falcon classes. Assume that all methods are **instance methods** (not static!).

The notation `Bird::gulgate(Bird)` specifies a method called `gulgate` with parameter of type `Bird` from the `Bird` class.

- F1. The `Bird::gulgate(Bird)` method exists.¹
- F2. The `Bird::gulgate(Falcon)` method exists.
- F3. The `Falcon::gulgate(Bird)` method exists.
- F4. The `Falcon::gulgate(Falcon)` method exists.

a) Suppose we make a call to `bird.gulgate(bird);`

Which features are sufficient **ALONE** for this call to compile? For example if feature F3 or feature F4 alone will allow this call to compile, fill in the F3 and F4 boxes.

F1 F2 F3 F4 Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method. For example, if having features F2 and F4 only (and not F1 or F3) would result in `Bird::gulgate(Bird)` being executed, check boxes F2 and F4 only.

F1 F2 F3 F4 Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

F1 F2 F3 F4 Impossible

b) Suppose we make a call to `falcon.gulgate(falcon);`

Which features are sufficient **ALONE** for this call to compile?

F1 F2 F3 F4 Impossible

Select a set of features such that this call executes the `Bird::gulgate(Bird)` method.

F1 F2 F3 F4 Impossible

Select a set of features such that this call executes the `Bird::gulgate(Falcon)` method.

F1 F2 F3 F4 Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Bird)` method.

F1 F2 F3 F4 Impossible

Select a set of features such that this call executes the `Falcon::gulgate(Falcon)` method.

F1 F2 F3 F4 Impossible

¹ In other words, the Bird class has a method with the signature `gulgate(Bird)`

5. Hofferly (8 Points). Suppose we have the following generic HOF interface:

```
public interface UnaryFunction<T> {
    public T apply(T x);
}
```

A spy is a `UnaryFunction` that wraps around another `UnaryFunction`, `f`. For any input `x`, it returns the result `f` would return, but it remembers the arguments on which it has been called and can print them out on command. For example, if `SquareFunction` represents a function that squares integers:

```
UnaryFunction<Integer> sq = new SquareFunction();
System.out.println(sq.apply(4)); // prints "16", not including quotes
```

```
Spy<Integer> spy = new Spy<>(sq);
System.out.println(spy.apply(5)); // prints "25", not including quotes
System.out.println(spy.apply(2)); // prints "4"
System.out.println(spy.apply(3)); // prints "9"
spy.printArgumentHistory(); // prints "5 2 3 "           non-destructive!
spy.printArgumentHistory(); // prints "5 2 3 "           and not including quotes
```

Complete the `Spy` class below. You may assume you have access to a working `ArrayDeque` or `LinkedListDeque`, if necessary. **Only write one statement per line.**

```
public class Spy<T> implements UnaryFunction<T> {
    ArrayDeque<T> history;
    UnaryFunction<T> function;

    public Spy(UnaryFunction<T> function) {
        this.history = new ArrayDeque<>();
        this.function = function;
    }

    @Override
    public T apply(T x) {
        history.addLast(x);
        return function.apply(x);
    }

    public void printArgumentHistory() {
        history.printDeque();
    }
}
```

```
interface Deque<Item>
void addFirst(Item x);
void addLast(Item x);
boolean isEmpty();
int size();
void printDeque();
Item get(int index);
Item removeFirst();
Item removeLast();
```

Login: _____

6. Skippify (12 points).

a) (9 points). Suppose we have the following `IntList` class, as defined in lecture and lab, with an added `skippify` function. **Only write one statement per line.**

```
public class IntList {
    public int first;
    public IntList rest;

    public static IntList list(int... args) { ... }

    @Override
    public boolean equals(Object o) { ... }

    public void skippify() {
        IntList p = this;
        int n = 1;
        while (p != null) {
            IntList next = p.rest;
            for (int i = 0; i < n; i += 1) {
                if (next == null) {
                    return; // NOTE: This should be a break to make more sense.
                }
                next = next.rest;
            }
            p.rest = next;
            p = p.rest;
            n++;
        }
    }
    ...
} // Reminder: Only write one statement per line.
```

Suppose that we define two `IntLists` as follows.

```
IntList A = IntList.list(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
IntList B = IntList.list(9, 8, 7, 6, 5, 4, 3, 2, 1);
```

Fill in the method `skippify` such that the result of calling `skippify` on A and B are as below:

After calling `A.skippify()`, A: (1, 3, 6, 10)

After calling `B.skippify()`, B: (9, 7, 4)

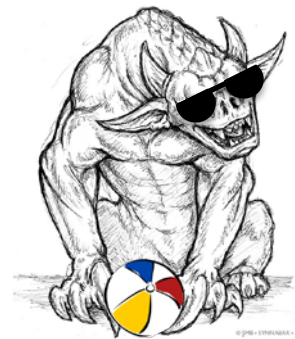
b) (3 points). Write a JUnit test that verifies that `skippify` is working correctly on the two lists above. You may not need all lines.

```
@Test
public testSkippify() {
    IntList A = IntList.list(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    IntList B = IntList.list(9, 8, 7, 6, 5, 4, 3, 2, 1);
    IntList expectedA = IntList.list(1, 3, 6, 10);
    IntList expectedB = IntList.list(9, 7, 4);
    A.skippify();
    B.skippify();
    assertEquals(expectedA, A);
    assertEquals(expectedB, B);
}
```

7. (0 Points). The present day Panama Canal was originally slated to be built in Nicaragua by a vote of 308 to 2 in the House of Representatives in 1902. However, intense lobbying efforts by consultant William Cromwell and Senator Mark Hanna resulted in the project being moved to Panama. One of their key lobbying strategies was to showcase a Nicaraguan postage stamp featuring what?

The Momotombo Volcano “in magnificent eruption.”

Warning: Official Grigometh Relaxation Area.



Login: _____

8. Dedup (8 points). Fill in the blanks below to correctly implement `removeDuplicates`.

```
public class IntList {

    public int first;
    public IntList rest;

    public IntList (int f, IntList r) {
        this.first = f;
        this.rest = r;
    }

    /**
     * Given a sorted linked list of items - remove duplicates.
     * For example given 1 -> 2 -> 2 -> 2 -> 3,
     * Mutate it to become 1 -> 2 -> 3 (destructively)
     */
    public static void removeDuplicates(IntList p) {
        if (p == null) {
            return;
        }
        IntList current = p.rest;
        IntList previous = p;
        while (current != null) {
            if (current.first == previous.first) {
                previous.rest = current.rest;
            } else {
                previous = current;
            }
            current = current.rest;
        }
    }
}
```

9. Interfaces (8 points). On project 1B, you implemented the Deque interface as ArrayDeque and LinkedListDeque. In this problem, you'll improve your Deque interface by adding a new default method.

Your job: Add a default method `remove(Item x)` that removes **all items** equal to `x`. This method should return `true` if anything was removed. It should return `false` if nothing was removed. You may not use the `new` keyword. **Only write one statement per line. You may not need all the lines.** If you need to determine that two Items are equal, use the `.equals` method, i.e. don't use `==` to compare Items.

```
public interface Deque<Item> {
    void addFirst(Item x);
    boolean isEmpty();
    void printDeque();
    Item removeFirst();
    void addLast(Item x);
    int size();
    Item get(int index);
    Item removeLast();
    default boolean remove(Item x) {
        int length = size();
        for (int i = 0; i < length; i++) {
            Item first = removeFirst();
            if (!(first != null && first.equals(x)) || first == x) {
                addLast(first);
            }
        }
        return length != size();
    }
}
/* Answers using the new keyword will not be given credit. */
```

For example, the test below should pass:

```
Deque<Dog> dd = new ArrayDeque<Dog>();
dd.addLast(new Dog("Ljilja"));
dd.addLast(new Dog("Rikhard"));
dd.addLast(new Dog("Spartacus"));
dd.addLast(new Dog("Rikhard"));
assertEquals(4, dd.size());
boolean rikhardRemoved = dd.remove("Rikhard");
assertTrue(rikhardRemoved);
assertEquals(2, dd.size());
```

Reminder: **You may not use the new keyword!** This is not just an arbitrary restriction, but ensures that we don't use any unnecessary space.

Login: _____

10. Grid (14 points). Consider an array implementation of an abstract data type for a grid of numbers, defined as follows:

```
public class ArrayGrid implements Grid {
    private int[][] nums;
    private int rows;    /* Number of rows in the Grid. */
    private int cols;    /* Number of cols in the Grid. */

    /** Creates a number grid initially full of zeroes, where r and c > 0.
     *  * nums is created with extra space to avoid immediate resizing. */
    public ArrayGrid(int r, int c) {
        nums = new int[r * 2][c * 2];
        rows = r;  cols = c;
    }

    /** Resizes the underlying nums array to be r arrays of length c each. */
    private void resize(int r, int c) { /* Code not shown */ }

    /** Gets a copy of the cth column. */
    private int[] getColCopy(int c) { /* Code not shown */ }

    @Override
    /** Prints out the grid. */
    public void print() { /* Code not shown */ }
}
```

For example, if we call `Grid ng = new ArrayGrid(3, 3)`, then call `ng.print()`, we'd get:

```
0 0 0
0 0 0
0 0 0
```

a) **(5 points)** Fill in the `addRow` method below. Assume that `row` is the same length as `cols`. **Only write one statement per line. You may not need all the lines.**

```
@Override
public void addRow(int[] row) {
    if (rows >= nums.length) {
        resize(nums.length * 2, nums[0].length);
    }
    System.arraycopy(row, 0, nums[rows], 0, cols);
    rows++;
    _____
    _____
    _____
}
```

- b) (9 points) Fill in the rotateColumn method below. Assume that $0 \leq c \leq \text{cols} - 1$. Assume that $0 \leq x \leq \text{rows} - 1$. You may not need all lines. You may only write one statement per line.

```

@Override
/** Rotate column c (only column c) by x positions downwards. */
public void rotateColumn(int c, int x) {
    int[] copy = getColCopy(c);
    for (int i = 0; i < rows; i++) {
        nums[(i + x) % rows][c] = copy[i];
    }
    _____
    _____
    _____
    _____
    _____
    _____
}

```

For example, the code on the left yields the outputs on the right:

1: Grid ng = new ArrayGrid(3, 3)	0 0 0 1	0 0 6 2
2: ng.addCol(new int[]{1, 2, 3});	0 0 0 2	0 0 0 3
3: ng.addRow(new int[]{4, 5, 6, 7});	0 0 0 3	0 0 0 7
4: ng.print();	4 5 6 7	4 5 0 1
5: ng.rotateColumn(2, 1);		
6: ng.rotateColumn(3, 3);	Line 4 output	Line 7 output
7: ng.print();		