CS 61BL Data Structures & Programming Methodology

Summer 2019

MIDTERM 1 SOLUTION

This exam has 8 questions worth a total of 30 points and is to be completed in 110 minutes. The exam is closed book except for one double-sided, handwritten cheat sheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided.

Write the statement 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."

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

Signature: Matthew Sit

Question	Points
1	2
2	3
3	0
4	5
5	5
6	4
7	4
8	7
Total	30

Name	Matthew Sit
Student ID	1234567890
GitHub account #	su19 - s1
Lab Section #	0 0 1
Name of person to left	Jackson Leisure
Name of person to right	Christine Zhou

- There may be partial credit for incomplete answers. Write as much of the solution as you can, but we may deduct points if your answers are much more complicated than necessary.
- **Work through the problems 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, and **you may not need all lines**. For code-writing questions, **write only one statement per line** and **do not write outside the lines.**
- Unless otherwise stated, all given code on this exam should compile. All code has been compiled and executed, but in the event that we do 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.'**

1. (2 pts) **Box**

   Fill in the comments in the main method to indicate what is outputted by each print line command.

```
public class Box {
    char[] name; int value; Box box;

    public Box(char[] n, int v, Box b) {
        name = n;
        value = v;
        box = b;
    }

    void update0(int value) {
        this.value = value + 1;
    }

    void update1(int value) {
        value = this.value + 2;
    }

    void update2(int value) {
        value = value + 4;
    }

    void update3(char[] boxName) {
        name[2] = 'T';
        boxName = new char[]{'U', 'P', 'D', 'A', 'T', 'E', 'D'};
    }

    int update4(int value) {
        return value + 8;
    }

    void update5(Box box) {
        box.value = box.value + 16;
    }

    Box update6(Box box) {
        box = new Box(box.name, box.value, box.box);
        return box;
    }

    // ... continued on next page
```

```
    // ... continued from previous page

    public static void main(String[] args) {
        Box box = new Box(new char[]{'B', 'O', 'X'}, 0, null);
        int a = 0; int b = 0; int c = 0;
        box.update0(a);
        box.update1(b);
        box.update2(c);
        System.out.println(a + b + c); // 0

        Box other = new Box(new char[]{'O', 'T', 'H', 'E', 'R'}, 2, box);
        System.out.println(other.name); // this println outputs: OTHER
        box.update3(other.name);
        System.out.println(box.name); // BOT

        System.out.println(other.name); // OTHER

        other.update4(box.value);
        other.update5(box);
        System.out.println(box.value); // 17

        System.out.println(other.value); // 2

        // Write either true or false in the blanks below; do not abbreviate.
        Box result = box.update6(box);
        System.out.println(result == box); // false

        System.out.println(result.name == box.name); // true

        System.out.println(result.value == box.value); // true

        System.out.println(result.box == box.box); // true

        System.out.println(box == other.box); // true
    }
}
```

2. (3 pts) **I'll Have My Cereal Non-Static Please**

   (a) Fill in the comments in the main method to indicate what is outputted by each print line command. If there is an error, write **CE** in the blank to indicate a compiler error, and **RE** to indicate runtime error.

```
public class Cereal {
    int numMarshmallows;
    static double crunchiness;

    public Cereal() {
        numMarshmallows = 100;
        crunchiness = 1.0;
    }

    static void pour() {
        crunchiness -= 0.1;
    }

    void eat(int quantity) {
        numMarshmallows -= quantity;
        if (quantity > 10) {
            crunchiness -= 0.05;
        }
    }

    // ... continued on next page
```

```
    // ... continued from previous page

    public static void main(String[] args) {
        Cereal a = new Cereal();

        a.pour();

        System.out.println(a.numMarshmallows); // 100

        System.out.println(a.crunchiness); // 0.9

        System.out.println(Cereal.crunchiness); // 0.9

        Cereal b = new Cereal();

        a.pour();

        System.out.println(a.crunchiness); // 0.9

        System.out.println(b.crunchiness); // 0.9

        b.eat(5);

        System.out.println(a.numMarshmallows); // 100

        System.out.println(b.numMarshmallows); // 95

        a.eat(15);

        System.out.println(a.numMarshmallows); // 85

        System.out.println(b.numMarshmallows); // 95

        System.out.println(a.crunchiness); // 0.85

        System.out.println(b.crunchiness); // 0.85
    }
}
```

(b) Suppose we add the following method to our `Cereal` class and call it from the main method as shown:

```
public class Cereal {
    ...

    static void devour() {
        numMarshmallows = 0;
    }

    ...

    public static void main(String[] args) {
        ...
        Cereal.devour();
    }
}
```

What will happen when we compile and run the code?
(Bubble in one choice.)

- ● Compiler error.
- ○ Nothing happens.
- ○ `this.numMarshmallows` will be set to `0`.
- ○ All instances' `numMarshmallows` will be set to `0`.

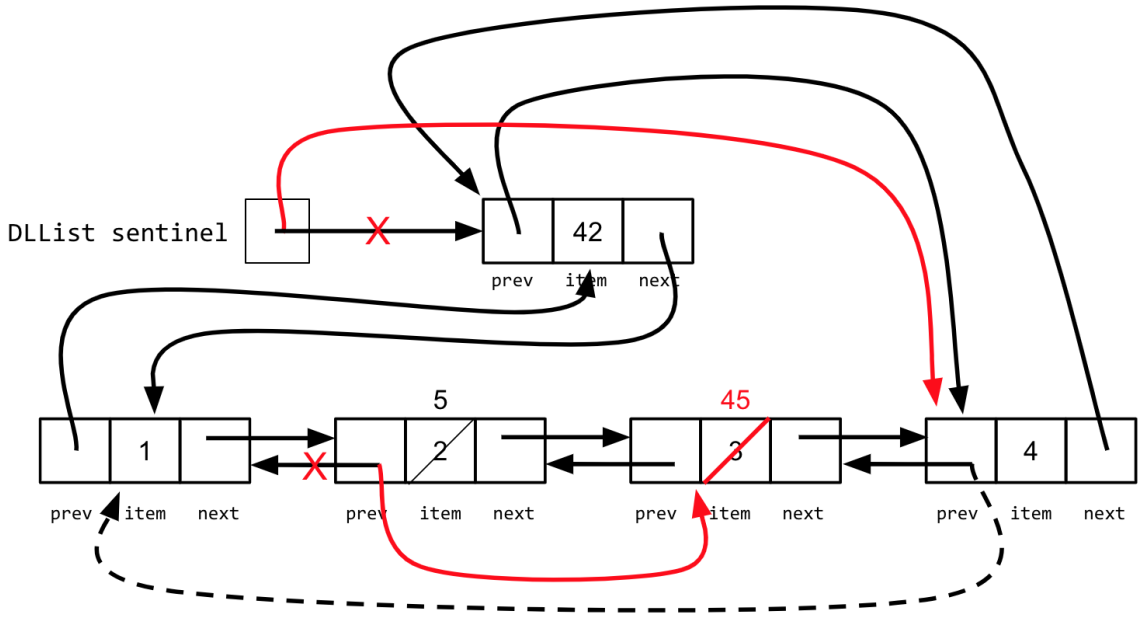
3. (0 pts) **PNH**

This is a classic Google interview question: How many golf balls can fit in a school bus?

Click here to see an article explaining how you would answer this.

Name: Matthew Sit

4. **Double Trouble**



(a) (2 pts) Write a concise line of code that will change the `2` to a `5` as shown in the diagram above.

sentinel.next.next.item = 5;

Write a concise line of code that will change the `prev` of the node containing `4` from the solid arrow to the dashed arrow.

sentinel.prev.prev = sentinel.next;

(b) (3 pts) Modify the diagram above to illustrate the effects of the following lines of code. If a value or reference should no longer exist, cross it out and replace it by writing the new value near it or drawing a new arrow. Make your marks clearly. If you make a mistake, erase it completely. If you only have a pen, draft your answer on scratch paper first. (It turns out it doesn't matter whether the changes from part a are kept or not, since the below code doesn't rely on those references/values.)

```
sentinel.next.next.prev = sentinel.next.next.next;
sentinel.next.next.next.item += sentinel.prev.next.item;
sentinel = sentinel.prev;
```

5. (5 pts) ~~**Chipotle**~~ **Runtime Analysis** For each problem, **give the best and worst-case runtimes in terms of *N***, the input into each function. Your answer should be simple with no unnecessary leading constants or summations. Write your final answer in the provided blanks. If you make a mistake, do not cross your answer and write near the blank, please erase it and replace the answer in the blank.

(a) `tortillaOnTheSide` **Best Case**: $\Theta($*log(N)*$)$ **Worst Case**: $\Theta($*log(N)*$)$

```
private static int tortillaOnTheSide(int n) {
    int ret = 1;
    if (n <= 1024) {
        for (int i = 0; i < n * n; i++) {
            ret++;
        }
        return ret;
    }
    ret += tortillaOnTheSide(n / 2);
    return ret;
}
```

(b) `riceAndBeans` **Best Case**: $\Theta(2^N)$ **Worst Case**: $\Theta(3^N)$

```
// Math.random() returns a double between 0 and 1
private static int riceAndBeans(int n) {
    if (n == 0) {
        return 1;
    }
    int ret = riceAndBeans(n - 1) + riceAndBeans(n - 1);
    if (Math.random() > 0.99) {
        ret += riceAndBeans(n - 1);
    }
    return ret;
}
```

(c) fajitas **Best Case**: $\Theta(N)$ **Worst Case**: $\Theta(N)$

```
private static int fajitas(int n) {
    int ret = 1;
    if (n == 0) {
        return ret;
    } else if (n < 100) {
        for (int i = 0; i < n; i += 1) {
            ret += fajitas(i);
        }
        return ret;
    } else if (n == 100) {
        return ret;
    } else {
        return ret + fajitas(n - 1);
    }
}
```

(d) salsa **Best Case**: $\Theta(N)$ **Worst Case**: $\Theta(N)$

```
private static int salsa(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result += 1;
        }
        n /= 2;
    }
    return result;
}
```

(e) chipsAndGuac **Best Case**: $\Theta(log(N))$ **Worst Case**: $\Theta(log(N))$

```
private static int chipsAndGuac(int n) {
    int result = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            result += 1;
            n /= 2;
        }
    }
    return result;
}
```

6. (4 pts) **Max Pooling**

In this question, you will implement a function for performing max pooling, which is a technique used in deep learning for reducing the resolution of an image being fed through the layers of a neural network.

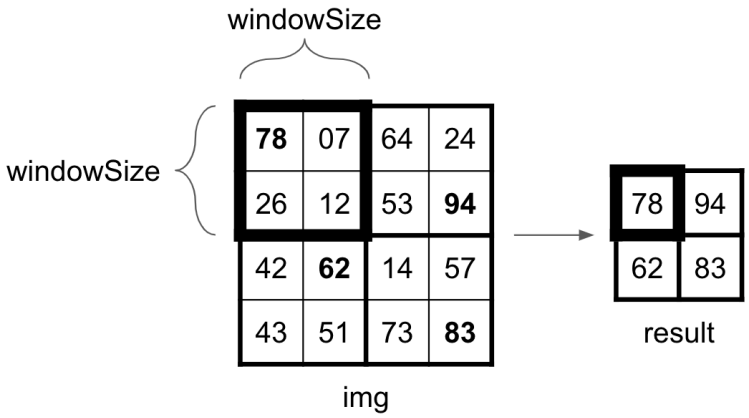
Images can be represented by two-dimensional arrays; the first dimension represents the rows of the image, and the second dimension is to represent the value of the pixel at a particular column index within a row.

For example, given a 2-dimensional image `int[][]` array called `img`, the pixel in the 5th row, and the 7th column can be accessed via `img[5][7]`.

One way we can implement max pooling is by cutting up our image (`img`) into small equal sized squares, and taking only the largest pixel value in each piece as the representative for that piece in our final downsampled image.

You may assume that the number of rows in the `img` and the number of columns in the image are both divisible by `windowSize` (the height/width of our square window), although you should not assume that `img` has the same number of rows as columns. However, all the rows are of the same length (they have the same number of columns). You may also assume that all pixel values in `img` are greater than or equal to `0`, that `img` is a properly instantiated rectangular 2-D array, and that there are no null values or invalid inputs to consider.

For example, this 4 by 4 `img` can be broken down into 4 equal-sized pieces of shape `windowSize` by `windowSize` (where `windowSize` is 2). The largest value in each of these pieces is recorded in `result` in their corresponding locations.



Complete the code block on the next page...

Name: Matthew Sit

Fill in the blanks so that the function behaves as intended.
(Hint: In Java, an `int` divided by an `int` will always result in a ...?)

```
static int[][] maxPool(int[][] img, int windowSize) {

        // resRows are the number of rows in result.
        // resCols are the number of columns in result.

        int resRows = img.length / windowSize;

        int resCols = img[0].length / windowSize;

        int[][] result = new int[resRows][resCols];

        for (int r = 0      ; r < img.length                    ; r++       ) {

            for (int c = 0      ; c < img[0].length                 ; c++       ) {

                // Java's Math.max() function only accepts two arguments at a time.
                // (Put one on the first line, and the second on the line below it).

                int largestSoFar = Math.max(img[r][c]               ,

                    result[r / windowSize][c / windowSize]          );

                result[r / windowSize][c / windowSize]          = largestSoFar;
            }
        }
        return result;
    }
```

7. (4 pts) **Whale whale whale...**

```
public class Mammal {
    /* (1) */ void migrate() {}
    /* (2) */ void splash(Mammal m) { }
}
public class Whale extends Mammal {
    /* (3) */ void migrate() { splash(this); }
    /* (4) */ void splash(Whale w) {}
}
public class Beluga extends Whale {
    /* (5) */ void migrate() {}
    /* (6) */ void splash(Mammal m) { m.migrate(); }
    /* (7) */ void splash(Beluga b) {}
}
public class Ocean { public static void main(String[] args) {
    Beluga b = new Beluga();
    Mammal mb = new Beluga();
    Mammal mw = new Whale();
    Whale wb = new Beluga();
    /* line goes here */
}}
```

For each of the following scenarios, determine what happens if the line was added to the end of the main method and put an 'X' in 'CE' (compiler error), 'RE' (runtime error), or the number(s) for the corresponding function(s) that run. **If multiple functions are run, mark them all.** If some functions run and it leads to a runtime error, mark the ones that successfully run and 'RE'.

	CE	RE	1	2	3	4	5	6	7
1. `b.migrate();`							X		
2. `mb.migrate();`							X		
3. `mw.migrate();`					X	X			
4. `b.splash(b);`									X
5. `b.splash(mb);`							X	X	
6. `b.splash(mw);`					X	X		X	
7. `((Beluga) mw).splash(b);`		X							
8. `((Whale) mb).splash(b);`						X			
9. `((Mammal) wb).splash(wb);`							X	X	

8. **Another Enigma**

There may be many possible ways of storing incoming data. We saw this in Project 0: Enigma, when you had to make a design decision on how to store your `cycles` String in `Permutation.java`. Here, we will represent those `cycles` using non-encapsulated linked lists.

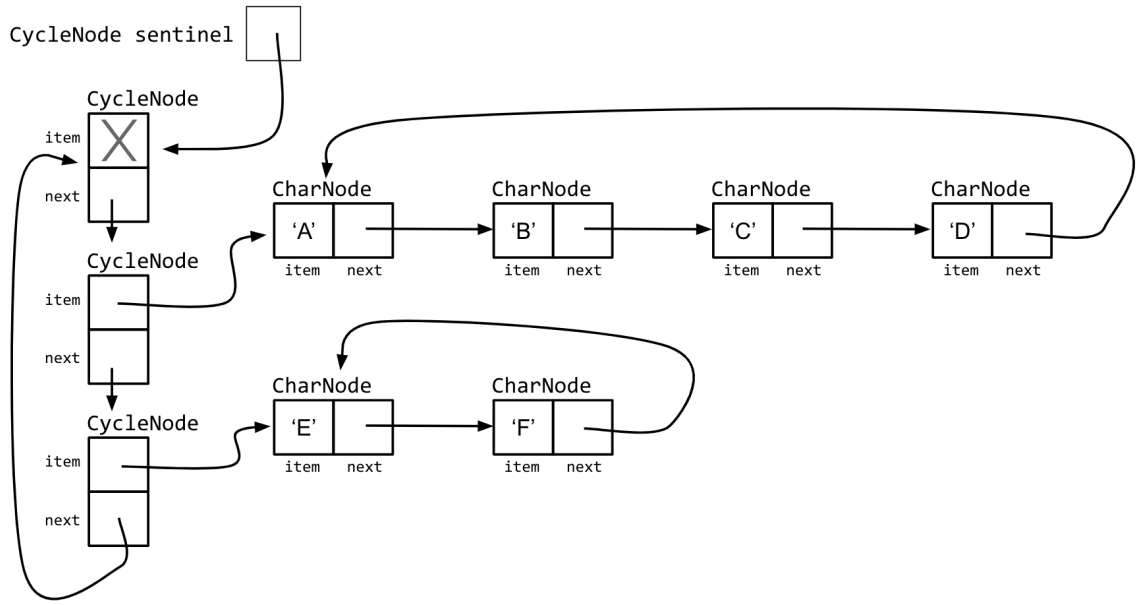
Recall that `String cycles` is of the format exemplified by something like `(ABCD)(EF)`. In that example, `A` maps/permutes to `B`, `B` to `C`, `D` to `A`, etc. Also recall that if a character is not specified by the cycles, it maps to itself.

For this problem, you may assume that we will only ever encounter capital letters that are in the Alphabet (as well as the parentheses), and that there are no space characters. You may also assume that input will always be of the correct format.

```
/** CycleNode is a linked list node representing a single cycle group of the
    cycles. */
public class CycleNode {
    CharNode item;
    CycleNode next;

    public CycleNode(CharNode item, CycleNode next) {
        this.item = item;
        this.next = next;
    }
}

/** CharNode is a linked list node representing a single character in a cycle. */
public class CharNode {
    char item;
    CharNode next;

    public CharNode(char item, CharNode next) {
        this.item = item;
        this.next = next;
    }
}
```

On the next page, you will see an example of what is described here, and on the following pages, you will fill in the skeleton code to implement the constructor of `Permutation.java`, as well as its character `permute` method.

This is what our data structure should look like after the constructor is called with the input argument `String cycles` of `(ABCD)(EF)`.



There is nothing to do on this page, proceed to the next pages to implement this.

(You may use the below area for scratch work, but make sure to move your final answers to the appropriate areas as this page will not be graded.)

(a) (4 pts) Fill in the blanks for the constructor. Each cycle should be represented as a cyclic chain of `CharNodes`, where each node holds a character and points at the next node it should map to. One `CharNode` of each of these cycles should be held as the `item` of a `CycleNode`, and each `CycleNode` should point at the next cycle node. We will use a `sentinel` for the `CycleNodes`, but not for the `CharNodes`.

```
public class Permutation {
    private CycleNode sentinel;

    public Permutation(String cycle) {

        sentinel = new CycleNode(null, null);

        sentinel.next = sentinel;

        CycleNode currCycle = sentinel;

        // Gets the letters of one cycle at a time, providing them as cyc.
        for (String cyc : cycle.split("\\)\\(")) {
            // Remove leading/trailing parentheses.
            cyc = cyc.replaceAll("\\(", "");
            cyc = cyc.replaceAll("\\)", "");
            // cyc is now just the letters of a single cycle.

            CharNode head = new CharNode(cyc.charAt(0), null);

            CharNode currCharNode = head;

            for (int i = 1      ; i < cyc.length()      ; i++          ) {

                currCharNode.next = new CharNode(cyc.charAt(i), null);

                currCharNode = currCharNode.next;
            }

            currCharNode.next = head;

            currCycle.next = new CycleNode(head, sentinel);

            currCycle = currCycle.next;
        }
    }
}
```

(b) (3 pts) Fill in the blanks for the character permute function. Given a character argument, `permute` should return the character that it maps to, according to the cycles provided via the constructor. You may assume that the constructor is properly implemented.

```
public class Permutation {

    private CycleNode sentinel;
    public Permutation(String cycle) { /* Implemented in part a. */ }

    public char permute(char c) {
        CycleNode currCycle = sentinel.next;

        while (currCycle != sentinel) {
            CharNode n = currCycle.item;

            while (true) {

                if (n.item == c                                    ) {

                    return n.next.item;                            

                } else {

                    n = n.next;                                    

                    if (n == currCycle.item                                    ) {
                        break;
                    }
                }
            }

            currCycle = currCycle.next;                            
        }

        return c;                                                  
    }
}
```