Faculty of Arts & Sciences
Department of Computer Sciences
CMPS 212—Intermediate Programming with Data Structure

Name: _____

Student ID: _____

- You have 90 minutes to complete this exam.
- This exam is closed-book. You may not use any computing devices including calculators.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, the only abbreviations that *are* allowed for this exam are:
  - `S.o.p` for `System.out.print`, and
  - `S.o.pln` for `System.out.println`.
- You do not need to write `import` statements in your code.
- If you enter the room, you must turn in an exam before leaving the room.
- You must show your Student ID to a TA or instructor for your exam to be accepted.

Good luck!

Score summary: (for grader only)

Problem	Description	Earned	Max
1	Linked list Tracing		10
2	Linked List Programming		15
3	Recursive Programming		15
4	Stacks & Queues		15
5	Hash Table		10
6	Tree		20
7	Huffman Tree		15
TOTAL	**Total Points**		**100**

1. Linked list Tracing (10 points)

```
class StringNode {
   private String data;
   private StringNode link;


   public static String foo1(StringNode current) {
      if (current==null)
            return "";

      return foo1(current.link) + current.data;
   }


   public static String foo2(StringNode current) {
      if (current==null)
            return "";

      return current.data + foo2(current.link);
   }


  public static void main (String [] args) {
      // omitted code: build a linked list to contain the following nodes:
      // list = {"a", "b", "c", "d"} or list = "a"→"b"→"c"→"d"
      // the nodes are of type StringNode
      // list is the head of the linked list

      System.out.println("output1 = " + StringNode.foo1(list));
      System.out.println("output2 = " + StringNode.foo2(list));
  }
}
```

2. Linked List Programming (15 points)

Write a method **`removeMin()`** and add it to the `LinkedIntList` class. The method should remove the smallest element value from the linked list. Suppose a `LinkedIntList` variable named `list` stores the following elements:

```
[8, 4, 7, 2, 9, 4, 5, 3]
```

If you made the call of `list.removeMin();`, the list would then store the elements:

```
[8, 4, 7, 9, 4, 5, 3]
```

If the list has more than one occurrence of the minimum value, the first occurrence is the one that should be removed. For example, `[8, 10, 13, 8, 8, 14, 9, 8, 11]` becomes `[10, 13, 8, 8, 14, 9, 8, 11]`. If the list is empty, you should throw a `NoSuchElementException`.

For full credit, obey the following restrictions in your solution. A solution that disobeys them can get partial credit.

- Do not call any other methods on the `LinkedIntList` object, such as `add`, `remove`, or `size`.
- Do not create new `ListNode` objects (though you may have as many `ListNode` variables as you like).
- Do not use other data structures such as arrays, lists, queues, etc.
- Your solution should run in O(*N*) time, where *N* is the number of elements of the linked list.

3. Recursive Programming (15 points)

Write a recursive method called parenthesize that takes a String and an integer n as parameters and that prints the string inside n sets of parentheses. For example, this code:

```
parenthesize("Joe", 2);
System.out.println();                                    // to complete line of output
parenthesize("The American University of Beirut", 6);
System.out.println();                                    // to complete line of output
parenthesize("midterm", 1);
System.out.println();                                    // to complete line of output
```

should produce these 3 lines of output:

```
((Joe))
((((((The American University of Beirut))))))
(midterm)
```

Your method should throw an IllegalArgumentException if passed a negative number. It could be passed 0, as in:

```
parenthesize("CMPS 212, Summer 2012", 0);
System.out.println();  // to complete line of output
```

In this case the output would have no (i.e., 0) parentheses:

```
CMPS 212, Summer 2012
```

You are not allowed to construct any structured objects (no array, ArrayList, String, StringBuilder, etc) and you may not use a while loop, for loop, or do/while loop to solve this problem; you must use recursion.

4. Stacks & Queues (15 points)

Write a method `interleave` that accepts a queue of integers as a parameter and rearranges the elements by alternating the elements from the first half of the queue with those from the second half of the queue. For example, suppose a variable `q` stores the following sequence of values:

```
front [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] back
```

and we make the call of `interleave(q);`, the queue should store the following values after the call:

```
front [1, 6, 2, 7, 3, 8, 4, 9, 5, 10] back
```

To understand the result, consider the two halves of this list. The first half is `[1, 2, 3, 4, 5]` and the second half is `[6, 7, 8, 9, 10]`. These are combined in an alternating fashion to form a sequence of interleave pairs: the first values from each half (1 and 6), then the second values from each half (2 and 7), then the third values from each half (3 and 8), and so on. In each pair, the value from the first half appears before the value from the second half.

The previous example uses sequential integers to make the interleaving more obvious, but the same process can be applied to any sequence of even length. For example, if `q` had instead stored these values:

```
front [2, 8, -5, 19, 7, 3, 24, 42] back
```

Then the method would have rearranged the list to become:

```
front [2, 7, 8, 3, -5, 24, 19, 42] back
```

Your method should throw an `IllegalArgumentException` if the queue does not have even size. **You may use ONE STACK as auxiliary storage to solve this problem**. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You may not use recursion to solve this problem. For full credit, your solution must run in O(*n*) time, where *n* represents the size of the queue. Use the `Queue` interface and `Stack`/`LinkedList` classes discussed in lecture

5. Hash Table (10 points)

1) (5points) Draw the contents of the hash table after inserting 16, 6, 9, 5, 15, 7, 3 given that table size is 11, quadratic probing collision handling, and `h(key) = (key + probe * probe) mod tableSize;` `probe++;` (`probe` is initialy set to 0)

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Table1

2) (5 points) In the hash table Table1 of 1), show the indexes of the spots that are examined when searching for:

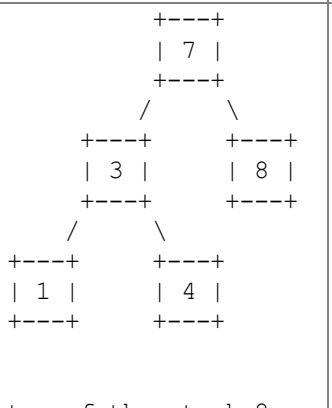
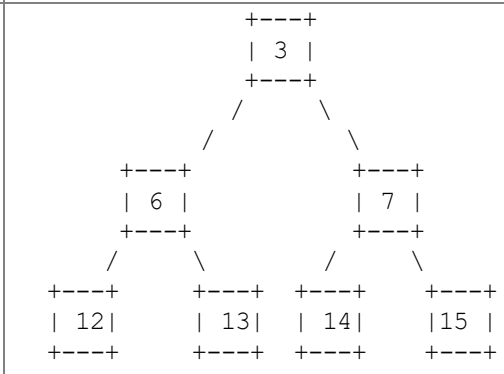
a. 4 Indexes:_____

b. 14 Indexes:_____

6. Tree (20 points)

1) The post-order traversal of a binary search tree gives: 20 35 30 41 45 42 40 53 58 60 55 52 .
   Draw a diagram of the tree. [10 points]

2) (10 points) Write a **recursive** method **`treeToStack`** that could be added to the **`IntTree`** class. **`treeToStack`** method should push all the data values of a tree into a **stack**. If the tree is a **binary search tree**, then the lowest data value should be at the bottom of the stack and the highest value should be at the top. **`treeToStack`** should return a **`Stack`** of **`Integer`**.

BST tree	**Binary tree**
 top of the stack=8 Bottom of the stack=1	 top of the stack=15 Bottom of the stack=12

You may define private helper methods to solve this problem, but otherwise you may not call any other methods of the **`IntTree`** class nor create any data structures such as arrays, lists, etc. Your method should not change the structure or contents of either of the two original trees being examined.

7. Huffman Tree (15 points)

Huffman coding is an example of how binary tree can be used to compress data. It compresses the data by reducing the number of bits that represent every character. In this problem, you are supposed provide a Huffman character-to- binary mapping table for the following text: SUSIE SAYS IT IS EASY. You have to show /sketch all the steps used to generate the Huffman tree then fill the table below.

Character	Huffman binary Code