COP 3530
Data Structures

Midsemester Exam

Name: ________________________________

March 8, 2007

This exam has 4 questions. Each question starts on a new page. Please answer each question on its page. You may assume `java.util` has been imported. There will be no deductions for lack of commenting. There will be no deductions for lack of import directives. There will be no deductions for minor syntax errors.

1. [**50 points**] `containsSum`, shown below, returns true if exactly two items in list `lst` sum to `K`.

```
public boolean containsSum( List<Integer> lst, int K )
{
    for( int i = 0; i < lst.size( ); i++ )
        for( int j = i + 1; j < lst.size( ); j++ )
            if( lst.get( i ) + lst.get( j ) == K )
                return true;

    return false;
}
```

   (a) What is the running time of `containsSum` when `lst` is an `ArrayList`?

   (b) What is the running time of `containsSum` when `lst` is a `LinkedList`?

   (c) Suppose it takes 4 seconds to run `containsSum` on a 10000-item `ArrayList`. How long will it take to run `containsSum` on a 50000-item `ArrayList`?

2. [**50 points**] This question requires that you implement some methods for a class that represents a doubly-linked list. In this question, **both a beginMarker and an endMarker are used**. You may assume an appropriate declared nested class `Node`. You may assume that the list does not store `null` values. You should only be following links; your solutions should not create or use any iterator classes.

   (a) Below you will implement `toString`, `removeFirst`, and `addLast`. Before writing the code, give the Big-Oh running time for each routine.

   (b) Implement `toString`. You may not invoke other methods of this class.

   ```
   public String toString( )
   {








   }
   ```

   (c) Implement `removeFirst` below. You may not invoke any other methods of the class. Be careful to correctly handle empty lists.

   ```
   public void removeFirst( )
   {







   }
   ```

   (d) Implement `addLast`. You may not invoke any other methods of the class.

   ```
   public void addLast( AnyType x )
   {





   }
   ```

   DID YOU REMEMBER TO GIVE THE BIG-OH?

3. [**50 points**] Many companies like to have phone numbers that can be spelled on the phone pad, rather than being random digits. For instance, 1-800-MATTRES is actually 1-800-628-8737. Suppose you are already assigned a phone number, and would like to see if there are any words that exist to match your number. This question asks you to write a function that returns a map in which the keys are phone numbers (as digits), and the values are lists of words that match that phone number.

   To simplify your code, you may assume the existance of the following method: `wordToNumber` takes a word, such as "REQUESTERS" and returns the keys on the phone touch pad that are associated with the letters in the word, in this case "7378378377". Thus, the phone phone number REQUESTERS is actually dialed with digits 737-837-8377. So is the phone number PERVERTERS. Again, **YOU MAY ASSUME THAT `wordToNumber` IS WRITTEN FOR YOU, AND HANDLES ALL ISSUES REGARDING UPPER/LOWER CASE CONVERSIONS.**

   Write method `groups` that takes an array of `String` as parameter and returns a `Map` is which each key is a phone number represented with digits, and the corresponding value is a list of words with that phone number. You may invoke `wordToNumber` to compute the digit-phone number for each string. For instance, if the input array contained:

```
[ government, perverters, requesters, sequesters, internment ]
```

   the map would contain

```
{ 7378378377=[perverters,requesters,sequesters], 4683766368=[government,internment]}
```

   Write this routine below, using Java 5.

4. [**50 points**]

An `Entry` is an object that represents either an `String`, or a list of other `Entry` objects. The `Entry` interface is shown below.

```
interface Entry
{
      // Return true if Entry repesents an integer; false otherwise
    boolean isString( );

      // Return the represented string, or throw an exception
      // if this Entry is representing a List of other entries.
    String getString( );

      // Return the represented List, or throw an exception
      // if this Entry is representing a String.
    List<Entry> getList( );
}
```

An example of an `Entry` is a file entry in Windows XP, in which the file entry is either a single file or a folder (and the folder can contain other files or more folders).

Implement the public driver method `expandEntry`, which will invoke a private recursive method that you must also implement. The public method accepts an `Entry` as a parameter and returns all the `String`s represented in the `Entry`. If the `Entry` is representing a single `String`, then the resulting `Set` will have size 1. Otherwise, `Entry` is representing a list of other `Entry`s, and you should recursively include the `String`s represented by those `Entry`s. (The logic stated above is most likely found in the private recursive routine).

YOU MAY NOT USE static DATA FIELDS. To simplify your code, you may assume that no `Entry` refers to itself, either directly or indirectly.

(a) Implement the public `expandEntry` below, and on the facing page, implement the private `expandEntry` whose signature is shown:

```
public static Set<String> expandEntry( Entry e )
{
// Implement here


}

// Implement on the facing page
private static void expandEntry( Entry e, Set<String> result )
```

(b) [**20 PTS EXTRA CREDIT, but no points unless your code in part (a) works almost perfectly**] After implementing your code above, rewrite both the public and private routine **on another page** so that it still works if an `Entry` refers to itself either directly, or indirectly. To do so, have the private recursive routine accept a third parameter that maintains a set that stores all the `Entry` objects that have been processed. Your routine can then avoid processing an `Entry` twice. You may assume that `Entry` objects have implemented any operations required to store a `Set<Entry>`.