COP 3530
Data Structures

Midsemester Exam

Name: _____

October 19, 2006

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**] Consider the following implementation of the `clear` method (which empties any collection). This implementation is identical to the one in the `java.util` API.

```
public abstract class AbstractCollection<AnyType> implements Collection<AnyType>
{
    public void clear( )
    {
        Iterator<AnyType> itr = this.iterator( );

        while( itr.hasNext( ) )
        {
            itr.next( );
            itr.remove( );
        }
    }
       ...
}
```

   (a) Suppose `LinkedList` extends `AbstractCollection` and does not override `clear`. What is the running time for `clear`?

   (b) Suppose `ArrayList` extends `AbstractCollection` and does not override `clear`. What is the running time for `clear`?

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

   (d) As clearly as possible, describe the behavior of this alternate implementation of `clear`:

```
public abstract class AbstractCollection<AnyType> implements Collection<AnyType>
{
    public void clear( )
    {
        for( AnyType item : this )
            this.remove( item );
    }
       ...
}
```

2. [**50 points**] This question requires that you implement some methods for a class that represents a doubly-linked list. In this question, **neither a beginMarker nor 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 may assume that the first node in the list is accessed by `first` and the last node is accessed by `last`, and if the list is empty, then both `first` and `last` are null. You should only be following links; your solutions should not create or use any iterator classes.

   (a) Below you will implement `toString`, `removeLast`, and `addFirst`. 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 `removeLast` below. You may not invoke any other methods of the class. Be careful to correctly handle empty lists and also lists with only one item.

```
public void removeLast( )
{









}
```

   (d) Implement `addFirst`. You may not invoke any other methods of the class. Make sure you have handled the special case of an empty list.

```
public void addFirst( AnyType x )
{









}
```

DID YOU REMEMBER TO GIVE THE BIG-OH?

3. [**50 points**] Assume that you have a `java.util.Map` in which the keys are `String`s and the values are `List<Integer>`s. The map represents words and the line numbers on which they occur.

   Write a routine, `linesToWords`, that returns a `Map` in which the keys are line numbers, and the values are lists of `String`s representing the words on the corresponding line numbers.

   For instance, if the map contains the four key/value pairs shown here:

   ```
   { hello=[2,3], good=[1,2], this=[1,5], if=[1,2,3] }
   ```

   then the map returned by `linesToWords` is

   ```
   { 1=["good","this","if"], 2=["hello","good","if"], 3=["hello","if"],5=["this"] }
   ```

   Write this routine below, using Java 5.

4. [**50 points**] Write a recursive routine to print a non-negative number in binary. You may assume that the only I/O routine that you have available is:

```
// Precondition: d is either 0 or 1
// Postcondition: d is printed
public static void printBinaryDigit( int d )
  { /* Implementation not shown */ }
```

YOU DO NOT HAVE TO IMPLEMENT `printBinaryDigit`.

Here are some examples (that also provide a clue to the algorithm):

```
N   Output
0    0
1    1
3    11
6    110
13   1101
26   11010
27   11011
```

Implement `printBinaryNumber` below:

```
// Precondition: n is not negative
// Postcondition: n is printed in binary
public static void printBinaryNumber( int n )
```