COP 3530
Data Structures

Midsemester Exam Version A

Name: _____

October 20, 2005

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**] Static method `removeHalf` returns the first half of a `List` (if there are an odd number of items, then slightly less than one-half of the list is removed). One possible implementation of `removeHalf` is shown below:

```
public static void removeHalf( List<?> list )
{
    int size = list.size( );

    for( int i = 0; i < size / 2; i++ )
        list.remove( 0 );
}
```

   (a) Why can't we use `list.size( ) / 2` as the test in the for loop?

   (b) Provide the Big-Oh running time, with a one-line explanation, if `list` is an `ArrayList`.

   (c) Provide the Big-Oh running time, with a one-line explanation, if `list` is a `LinkedList`.

   (d) Suppose we have two computers, Machine A and Machine B. Machine B is twice as fast as Machine A. How would the running time for `removeHalf` on Machine B compare to Machine A if Machine B is given an `ArrayList` that is twice as large as the `ArrayList` given to Machine A?

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, and there is a field that maintains the size of the list**; 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 shuld not create or use any iterator classes.

   (a) Implement `toString` and **PROVIDE ITS BIG-OH RUNNING TIME**. If you invoke other methods of this class, you must implement them.

```
public String toString( )
{









}
```

   (b) Implement the private method `addAfter` in the space shown below. `addAfter` places a new node in the list containing `x` after the node specified by `p`. If you invoke other methods, you must write those methods too.

```
private void addAfter( Node<AnyType> p, AnyType x )
{









}
```

   (c) Implement `addLast`.

```
public void addLast( AnyType x )
{



}
```

3. [**50 points**] Write routine `groupWords` that takes an array of `String` as its parameter, and returns a `Map` in which the keys are numbers representing the length of a `String` and the corresponding value is a `List` of all `String` of that length. The `Map` must be sorted by string length.

   For example, if `arr` is

   ```
   [hello,world,if,by,and,two]
   ```

   the `Map` that is returned is

   ```
   { 2=[if,by], 3=[and,two], 5=[hello,world] }
   ```

   Complete `groupWords`, which is started below:

   ```
   public static Map<Integer,List<String>> groupWords( String [ ] arr )
   ```

4. [**50 points**]

You are given a `Map` that contains an email mailing list address book. In the `Map`, the keys are aliases, and the corresponding values are lists consisting of email addresses and other aliases. An alias is guaranteed to not contain the `@` sign, and an email address is guaranteed to contain the `@` sign. An example of a `Map` is

```
{ faculty=[weiss@fiu.edu,shawg@fiu.edu],
  staff=[maureen@fiu.edu,canedah@cs.fiu.edu],
  facstaff=[faculty,staff],
  all=[facstaff,maidique@fiu.edu,rosenberg@fiu.edu,weiss@fiu.edu] }
```

Write routine `expandAlias` that takes a `Map` and an alias and returns the `Set` of all email address that it expands to. For example, expanding `all` yields a `Set` containing six email addresses.

Note that if the alias parameter is an email address, `expandAlias` returns a `Set` of size one. If the alias parameter is not an email address, but is an invalid alias (i.e. not in the map), you can return a `Set` of size zero.

To simplify your logic, you may assume that there are no cycles in which an alias eventually includes itself.

You can test if `@` is present in a `String` by using `String` method `indexOf` (which will return a negative number if the test fails, and a valid index otherwise).

You can also use `addAll` to add the entire contents of another `Set` into an existing `Set`. This can simplify your code.

Complete `expandAlias`, which is started below:

```
public Set<String> expandAlias( Map<String,List<String>> addressBook, String alias )
{
```