CMPUT 115 Section B3
Final Exam

April 23, 2001

Instructions:

- This is a closed book, no notes exam.
- You will be given 2 hours to complete the exam.
- Try to put all of your answers in the space provided.
- Be sure to write your student id number on each internal page.

First Name:

Last Name:

Student ID: _____

1. [4 Marks] Recall that:
   - `(new Integer(8)).hashCode()` returns `8`, and in general
   - `(new Integer(i)).hashCode()` returns `i`.

Consider the **Hashtable** in the diagram below (the left diagram). This Hashtable contains Integer objects (the keys) and String objects (the values). What would it look like after a call to the **rehash** method? Put your answer in the diagram on the right.

Before rehash

0	21, "A"
1	14, "B"
2	
3	17, "C"
4	32, "D"
5	
6	

After rehash

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

2. [1 Mark] Below are two fragments of code that differ only in the part that is bold. Which code fragment would you expect to execute **most quickly**?

a.
```
Queue q = new QueueVector();
for ( int i = 0; i < 20000 ; i++ ) q.add(new Integer(i));
while ( !q.isEmpty() ) q.remove();
```

b.
```
Queue q = new QueueList();
for ( int i = 0; i < 20000 ; i++ ) q.add(new Integer(i));
while ( !q.isEmpty() ) q.remove();
```

3. [1 Mark] Below are three fragments of code that differ only in the part that is bold. Which code fragment would you expect to execute **most quickly**?

a.
```
OrderedStructure os = new OrderedVector();
for ( int i = 10000; i > 0 ; i-- )
   os.add( new Integer(i) );
```

b.
```
OrderedStructure os = new OrderedList();
for ( int i = 10000; i > 0; i-- )
   os.add( new Integer(i) );
```

c.
```
OrderedStructure os = new BinarySearchTree();
for ( int i = 10000; i > 0; i-- )
   os.add( new Integer(i) );
```

4. [4 Marks] Draw the **BinarySearchTree** built from the following additions.

```
BinarySearchTree bst = new BinarySearchTree();
bst.add( "M" );
bst.add( "B" );
bst.add( "T" );
bst.add( "D" );
bst.add( "T" );
bst.add( "M" );
bst.add( "T" );
bst.add( "Z" );
```

5. [3 Marks] Redraw the tree you drew for question **4** as it would appear after the following removals.

```
bst.remove( "Z" );
bst.remove( "B" );
bst.remove( "M" );
```

6. [2 Mark] Given the tree you drew for question **4** (**not** question 5), what order will a **postorder** traversal return the elements in?

7. [2 Marks] Consider code that uses a Vector. There are two approaches to traversing the elements in the Vector, one that uses an Iterator and one that does not (see below). In what way is the **Iterator** approach superior?

```
// does not use Iterator
for ( int i = 0; i < data.size(); i++ )
  System.out.println( data.elementAt(i) );

// uses Iterator
Iterator it = data.elements();
while( it.hasMoreElements() )
  System.out.println( it.nextElement() );
```

8. [2 Marks] What is the time-complexity of the following code fragment? Express your answer as a function of **N** and clearly show how you arrive at your answer.

```
ChainedHashtable ht = new ChainedHashtable(3*N);
for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    ht.put( new Integer(i), new Integer(j) );
```

9. [5 Marks] The following is a partial implementation of the **ChainedHashtable** class from the structure package. A few lines of code are replaced with blanks (boxes actually). Each blank corresponds to one line of code. Fill in each of the blanks.

```
public class ChainedHashtable implements Dictionary
{
    protected List data[];
    protected int count;
    protected int capacity;

    public ChainedHashtable(int size)
    // pre: size > 0
    // post: constructs a new ChainedHashtable
    {
        data = new List[size];
        capacity = size;
        count = 0;
    }

    protected List locate(Object key)
    // post: returns list potentially containing key, if in table
    {
        int hash = Math.abs(                                    );
        if (data[hash] == null)
                                                                
        return                          
    }

    public Object remove(Object key)
    // pre: key is non-null object
    // post: removes key-value pair associated with key
    {
        List l = locate(key);
        Association pair =
                                                                
        if (pair == null) return null;
        count--;
        return                          
    }
}
```

Student ID: _____

10. [2 Mark] The **clear** method of the **ChainedHashtable** class from the structure package has a mistake (i.e. a bug) in it! What is the mistake?

```
public void clear()
//post: removes all elements from ChainedHashtable
{
  int i;
  for (i = 0; i < capacity; i++) {
    data[i].clear();
  }
  count = 0;
}
```

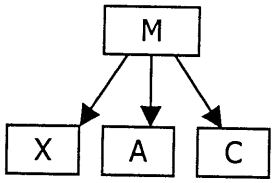
11. [4 Marks] Implement the **get** method of the **BinarySearchTree** class. In your get method, you may use the **locate** method.

```
public class BinarySearchTree implements OrderedStructure {
  protected BinaryTreeNode root;
  protected int count;

  public Object get(Object val)
  // pre: val is non-null
  // post: returns object found in tree, or null
  {

  }
}
```

Student ID: _____

Background information for questions 12 and 13.

A **ternary tree** is a tree in which each node has at most three children. One child is called the left child, one is called the middle child and the other is called the right child. Here is an example:



12. [1 Mark] Recall that the height of a tree is the length (i.e. the number of edges) of the longest path between the root node and a leaf. What is the height of the shortest ternary tree containing 45 nodes?

    a. 3

    b. 4

    c. 5

    d. 6

13. [4 Marks] The following is a partial implementation of the **TernaryTreeNode** class. Using recursion, implement a method named **countLeaves**. The countLeaves method should take a node and return the number of leaves in the sub-tree rooted at that node.

```
public class TernaryTreeNode {
  protected Object val; // value associated with node
  protected TernaryTreeNode parent; // parent of node
  protected TernaryTreeNode left;   // left child of node
  protected TernaryTreeNode middle; // middle child of node
  protected TernaryTreeNode right;  // right child of node

  public static int leaves(BinaryTreeNode n) {
  // post: returns the number of leaves in the subtree rooted at n
```