CMPUT 115 Section B1
Final Exam

April 18, 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.
- Please do not open the exam until you are instructed to do so.
- Be sure to write your student id number on each internal page.

First Name:

Last Name:

Student ID: _____

1. [5 Marks] Paying attention to time-efficiency, write a method that takes as an argument a Vector and returns a new Vector that contains all of the elements of the original Vector except that no duplicate values are included. The following is an example of how your code should work. There is room for your answer following the example.

```
Vector v1, v2;
v1 = new Vector();
v1.addElement( "c" );  v1.addElement( "b" );
v1.addElement( "c" );  v1.addElement( "c" );
v1.addElement( "b" );  v1.addElement( "a" );
// v1 now contains the String objects: c, b, c, c, b, a
v2 = copyWithoutDuplicates( v1 );
// v2 now contains the String objects: c, b, a
// and v1 should still contain the String objects: c, b, c, c, b, a

public static Vector copyWithoutDuplicates( Vector original )
// pre:  original is not null
// post: returns a Vector containing unique objects from original
// hint: consider which data structure could be used to temporarily
//       (and efficiently) store the objects
{
    Vector result = new Vector();




    return result;
}
```

2. [2 Marks] Draw the tree built by the following code fragment.

```
BinaryTreeNode root = new BinaryTreeNode( "a" );
root.setLeft( new BinaryTreeNode("b") );
BinaryTreeNode tmp = new BinaryTreeNode( "c" );
tmp.setLeft( new BinaryTreeNode("d") );
tmp.setRight( new BinaryTreeNode("e") );
root.setRight( tmp );
root.left().setRight( new BinaryTreeNode("f") );
```

3. [2 Marks] Consider the following **recursive** method.

```
public static void visit( BinaryTreeNode root )
{
  if ( root == null ) return;
  visit( root.right() );
  visit( root.left() );
  System.out.println( root.value() );
}
```

What output would be produced if the above method was called with the root of the tree you drew for question 2? (In other words what output would be produced by the call visit(root) where root is as built in question 2?)

4. [5 Marks] The following is a partial implementation of the **BinaryTreeNode** 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 BinaryTreeNode {
    protected Object val; // value associated with node
    protected BinaryTreeNode parent; // parent of node
    protected BinaryTreeNode left; // left child of node
    protected BinaryTreeNode right; // right child of node

    public void setLeft(BinaryTreeNode newLeft)
    // post: sets left subtree to newLeft
    //       re-parents newLeft if not null
    {
        if (left != null && (left.parent() == this))
            [                                        ]
        left = newLeft;
        if (left != null)
            [                                        ]
    }

    protected void setParent(BinaryTreeNode newParent)
    // post: re-parents this node to newParent
    {
        [                                        ]
    }

    public static int size(BinaryTreeNode n)
    // post: returns the size of the subtree rooted at n
    // hint: this method is recursive
    {
        if (n == null) return 0;
        return [                                        ]
    }

    public boolean isLeftChild()
    // post: returns true if this is a left child of parent.
    {
        if (parent() == null) return false;
       return [                                        ]
    }
}
```

5. [2 Marks] Why would the following method not make an effective hash function for the String class? (Hint: think of the effects on the efficiency of the major operations of the Hashtable class.)

```
public int hashCode()
{
    // returns the length of the String ...
    return this.length();
}
```

6. [4 Marks] Recall that:
   - (new Integer(1)).hashCode() would return 1,
   - (new Integer(2)).hashCode() would return 2, and in general
   - (new Integer(i)).hashCode() would return i.

   What output would the following code fragment produce? (Hint: think about where each key value pair will be placed.)

```
Hashtable ht = new Hashtable(11); // note capacity is 11

ht.put( new Integer(1), "some value" );
ht.put( new Integer(9), "some value" );
ht.put( new Integer(11), "some value" );
ht.put( new Integer(12), "some value" );
ht.put( new Integer(11), "some other value" );

Iterator it = ht.keys(); // get an iteration over the keys
while ( it.hasMoreElements() )
    System.out.println( it.nextElement() );
```

7. [5 Marks] Implement the **containsKey** method of the Hashtable class. You may use the locate method.

```
public class Hashtable implements Dictionary {
    protected static Association reserved =
                new Association("reserved",null);
    protected Association data[];
    protected int count;
    protected int capacity;

    protected int locate(Object key)
    // pre: key is non-null
    // post: returns ideal index of key in table
    {
        // You may assume that this is already implemented
    }

    public boolean containsKey(Object key)
    // pre: key is a non-null Object
    // post: returns true if key appears in hash table
    {




    }
}
```

8. [1 Mark] Integer objects 1, 2, 3, 4, 5, 6 and 7 are added to a **BinarySearchTree**. Which insertion order will produce the most efficient (i.e. shortest) tree?

a. 1, 2, 3, 4, 5, 6, 7

b. 7, 6, 5, 4, 3, 2, 1

c. 4, 3, 5, 2, 6, 1, 7

d. 4, 2, 6, 1, 3, 5, 7

9. [1 Mark] Below are three fragments of code that differ only in the part that is bold. Circle the fragment of code that would take the **least** amount of time to execute. (Hint: look at the order in which the elements are added.)

a.
```
OrderedStructure os = new OrderedVector();
for ( int i = 0; i < n; i++ )
   os.add( new Integer(i) );
```

b.
```
OrderedStructure os = new OrderedList();
for ( int i = 0; i < n; i++ )
   os.add( new Integer(i) );
```

c.
```
OrderedStructure os = new BinarySearchTree();
for ( int i = 0; i < n; i++ )
   os.add( new Integer(i) );
```

10. [2 Marks] How does the **load factor** of a Hashtable affect its efficiency?

11. [1 Mark] Assume that the variable **head** is a SinglyLinkedListElement that is at the head of a list with more than 2 elements. Which of the following code fragments correctly removes the second element from the list?

a.
```
head.setNext( head.next() );
```

b.
```
head.next() = head.next().next();
```

c.
```
head.setNext( head.next().next() );
```

12. [5 Marks] The following is a partial implementation of the **BinarySearchTree** class from the structure package. A few lines of code are replaced with blanks (boxes actually). Fill in these blanks. Each blank corresponds to one line of code. Recall that the locate method is recursive.

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

  protected BinaryTreeNode locate(BinaryTreeNode root,
                                  Comparable value)
  // pre: root and value are non-null
  // post: returned: 1) existing tree node with the desired value or
  //                 2) the node to which value should be added.
  {
        Comparable rootValue = (Comparable)root.value();
        BinaryTreeNode child;

        // found at root: done
        if (rootValue.equals(value)) return root;

        // decide which direction to look (left or right)
        if ( rootValue.compareTo(value) < 0 )
            child = [                                    ]
        else
            child = [                                    ]

        // if no child there: not in tree, return this node,
        if (child == null) return [          ]


        // else keep searching

        else return [                                    ]
  }

  protected BinaryTreeNode predecessor(BinaryTreeNode root)
  // pre: tree is not empty, root node has left child.
  // post: returns pointer to predecessor of root
  {
        BinaryTreeNode result = root.left();
        while (result.right() != null) {
            [                                    ]
        }
        return result;
    }
}
```