**44-542 Object Oriented Programming Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Exam 03 Part 2 (60 points) KEY** *please print*

1. (4 pts) Use the hash function **h(key) = key mod 10** to store records with keys 39, 77, 42, 49, 51, 32, and 50 in the hash table shown below. Use linear probing to resolve collisions. You must apply **h** to the keys in the order in which they are listed above.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **49** | **51** | **42** | **32** | **50** |  |  | **77** |  | **39** |

1. (5 pts) Draw a complete tree with eight nodes. Nodes should be empty. Do ***not*** insert values into any node. You are showing only the shape of the tree, and contents are irrelevant.



1. (5 pts) Draw a full tree of height 2. Nodes should be empty. Do ***not*** insert values into any node. You are showing only the shape of the tree, and contents are irrelevant.



1. (5 pts) For the binary search tree shown below, write the order in which nodes will be visited for a preorder traversal.



Answer:  **50 15 10 3 35 23 85 60 70 95**

1. (5 pts) Using the algorithm from class, add 17 to the binary search tree shown below. You do not need to re-draw the tree – just add the node in the appropriate position to the tree shown here.





1. (5 pts) Using the algorithm from class, remove 25 from the binary search tree shown below. Re-draw the tree in the space provided.



**DRAW TREE HERE**



1. (5 pts) Using the algorithm from class, perform a left rotation around 25 in the binary search tree shown below.



**DRAW TREE HERE**



1. (5 pts) Using the algorithm from class, insert 18 into the heap shown below.



**DRAW HEAP HERE**



1. (5 pts) Using the algorithm from class, remove the top element from the heap shown below.



**DRAW HEAP HERE**



1. (8 pts) Suppose we are writing a generic **PureStack** class, with an **ArrayList** as a private instance variable. The class header and the declaration of the private instance variable are shown below. The constructor creates a new empty array list; the push method uses the add method of the ArrayList class to push elements onto the stack. Add code for the **pop** method. The **pop** method is public and has no parameters; it removes and returns the top element on the stack. If the stack is empty, an **EmptyStackException** is thrown.

Note 1: Do not add any private instance variables to this class.

Note 2: Do ***not*** write code for the constructor or for the **push** method. The only code you will write is for the **pop** method.

**public class PureStack<E>**

**{**

**private ArrayList<E> stack;**

**public E pop()**

**{**

**if (stack.isEmpty())**

**{**

**throw new EmptyStackException();**

**}**

**return(stack.remove(stack.size() - 1));**

**}**

1. (8 pts) Suppose we are writing a generic **LinkedList** class. We have created a **Node** class as shown here:

**public class Node<E>**

**{**

**E data;**

**Node<E> nextNode;**

**public Node(E data)**

**{**

**this.data = data;**

**}**

**}**

The class header, constructor, and declaration of private instance variables are shown below. Add code for method **addFirst**. Method **addFirst** is public, does not return a value, and has one parameter of type **E** named **myObject**. This method creates a **Node** containing **myObject** as its content and inserts this node into the list, so that it is the first node in the list. It also increments **listLength** by 1.

You cannot add any private instance variables other than those shown here.

**public class ALinkedList<E>**

**{**

**private Node<E> listStart;**

**private int listLength;**

**public ALinkedList()**

**{**

**listStart = null;**

**listLength = 0;**

**}**

**public void addFirst(E myObject)**

**{**

**Node<E> newNode = new Node(myObject);**

**newNode.nextNode = listStart;**

**listStart = newNode;**

**listLength++;**

**}**