**C++ Activities and Lecture Notes – Chapters 12 and 13**

**Chapter 12 – Lists, Queues, and Stacks**

What is a stack?

What are the traditional names of the operations associated with stacks?

What is a queue?

What are the traditional names associated with queue operations?

What is a list?

What is a singly-linked list?

What is a doubly-linked list?

A standard stack problem: evaluation of a postfix expression

Consider the infix expression

4 \* ( ( 12 + 5 ) \* 3 ) + 7

What’s the outcome?

This same expression in postfix might be written as

12 5 + 3 \* 4 \* 7 +

One way to evaluate this expression is by using a stack.

[demonstration on the board follows…]

**EXERCISE:**

Convert the following expression to postfix and evaluate it

( 4 + ( 12 \* ( 4 + 3 ) \* 8 – 5 ) / 6 ) + 2

Now we have to be careful about non-commuting operations like – and /. Our convention will be that 4 5 / is the same thing as 4 / 5.

**Singly Linked List Implementation**

We could create a LinkedListNode as follows:

|  |
| --- |
| LinkedListNode |
| - LinkedListNode \* : next  - string : value |
| + string getValue()  + LinkedListNode \* getNext() |

Now a LinkedList object might be

|  |
| --- |
| LinkedList |
| - LinkedListNode \* head |
| + insert(string)  + remove(string) // remove first instance of the string encountered  + traverse() |

How would we then create and maintain a linked list?

**EXERCISE:**

Insert the following items into an empty LinkedList:

Batman, Superman, Aquaman, Wonder Woman, Green Lantern, Spider-Man, CatWoman

Now remove, in this order

Aquaman, Wonder Woman, Green Lantern

Our UML above is fairly abstract; we’ve specified no constructors and ignored the consequences of dynamic memory allocation. What additional methods would we need to implement this in C++?

How does the LinkedList destructor operate? Write the algorithm, then demonstrate it by deleting what remains of our list.

**Chapter 13 – Sets, Maps, and Priority Queues**

What is the distinction between a *sequence* and a *set*?

What is a *priority queue*?

What are some criteria that might cause one job to be served before another in a priority queue?

What is a *map*?

What is a *graph*?

What is a *binary search tree*?

A Binary Search Tree (BST) is a data structure with the following properties:

A node can have zero, one, or two children.

A node with no children is called a “leaf” node.

The two children are called “left” and “right.”

The data associated with the left child is strictly less than the data associated with the

parent.

The data associated with the right child is strictly greater than the data associated with

the parent.

There are no duplicate data items.

The left and right subtrees are also binary search trees.

The root of the tree has no parent, or its parent is a sentinel that serves to grant access to

the tree.

Binary search trees are fundamental data structures that serve as the parent classes to many other data structures. Sorting is performed automatically on insertion of data items and is preserved on deletion of data items. In-order traversal is very efficient. When height-balanced, the search, insert, and remove operations have O(lg n) performance.

**EXERCISE:**

Insert the following twelve items into a BST

55 23 98 12 5 88 34 24 90 30 99 40

Insertions always create a new leaf node.

**EXERCISES:**

Traverse the tree using the recursive algorithms VLR, LVR, and LRV. Which is in-order?

**Deleting nodes from a BST**

**If the node to be deleted is a leaf node:**

Delete the node.

**If the node to be deleted has only one child:**

Delete the node, and connect the child to the parent.

**If the node to be deleted has two children:**

Locate the in-order predecessor or successor of the node to be deleted.

Replace the data value of the node to be deleted with the data value of the

successor or predecessor node.

Delete the successor or predecessor node.

Note that the in-order predecessor is always is the rightmost child of the left subtree, and is a leaf node (no children). The in-order successor is always the leftmost child of the right subtree, and is also a leaf node.

**EXERCISE:**

Delete (in this order) 30, 12, 88, and 23 from the tree created in the previous exercise.

Show the state of the tree after every deletion.

Is our tree balanced? What does it mean to be balanced?

What is an *AVL Tree*?

The AVL tree (named after the inventors, G. M. Adelson-Velskii and E. M. Landis) requires an additional data item per node: the balance factor. The balance factor is the height of the right subtree minus the height of the left subtree. If the balance factor is negative, the tree is left-heavy; if positive, the tree is right-heavy. In a balanced tree the only balance factors present in any node are -1, 0, and +1.

**AVL Tree Rotations**

**RR Imbalance**

The node has a balance of +2, and its right child has a balance of +1 or 0.

A

B

C

Solution: perform a left rotation on A. B becomes the new root and A becomes its new left child

B

A C

**LL Imbalance**

The node has a balance of-2 and its left child has a balance of -1 or 0

C

B

A

Solution: Perform a right rotation on C. B becomes the new root and C becomes its new right child

B

A C

**RL Imbalance**

The node has a balance of +2 and the left child has a balance of -1

A

C

B

Solution: perform a right rotation on the right subtree, then a left rotation at the root.

A

B

C after right rotation on right subtree

B

A C after left rotation of root

**LR Imbalance**

The node has a balance of -2 and the right child has a balance of +1.

C

A

B

Solution: perform a left rotation on the left subtree, then a right rotation at the root.

C

B

A after left rotation of left subtree

B

A C after right rotation at the root

**EXERCISES**

Repeat the insertions from the first page into an initially empty AVL tree, maintaining balance as you go. Check using the demo at www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html.

<http://www.site.uottawa.ca/~stan/csi2514/applets/avl/BT.html>

Insert the integers 1-12 into an AVL tree.