Ryerson Polytechnic University - School of Computer Science
CPS305 - Data Structures
Final Exam - Fall '03

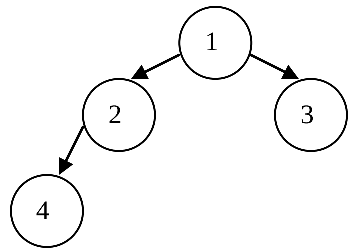
…

1. Answer the following questions in the space provided.
a. (4 marks) What is *clustering* in a hash table?
b. (4 marks) Name 2 advantages of a chained hash table over open addressing.
c. (4 marks) Define the term *binary search tree*.
d. (6 marks) Give the order of visiting the vertices of the given binary tree under

(a) preorder

(b) inorder

(c) postorder



2. (8 marks) Use the functions for Stacks and/or Queues developed in the text to write a function **Move(Stack *B, Stack *A)** that will do the following task:

Empty Stack A onto the top of Stack B in such a way that the entries that were in A keep the same relative order. An example is given below.

Important:

- Be sure to check for empty and full structures as appropriate.
- Move must be implementation independent i.e., nothing in Move may rely on the underlying implementation of the Stacks and/or Queues.
- You must access the Stacks and/or Queues **only** through their ADT functions.

Example:

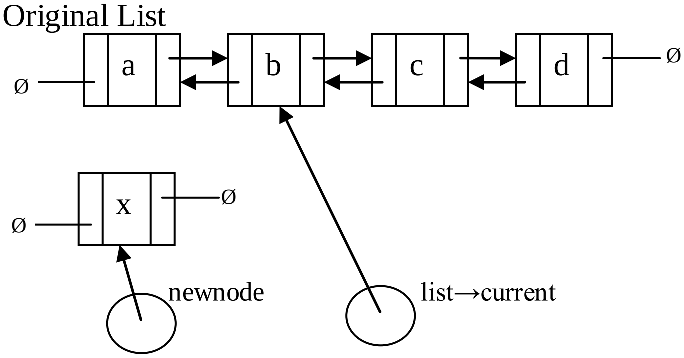
B before Move	**A** before Move	**B** after Move
a	1	1
b	2	2
c	3	3
		a
		b
		c

3. (8 marks) Consider doubly linked lists as described in your text. Declarations are given below. Consider the Original List below (note that ø denotes NULL). **Write a chunk of code** (NOT a complete function) that inserts newnode into the list between nodes for "b" and "c", so that the resulting list looks like the Modified List below.
Important: Your code may access the list **ONLY** through the variables: list→current, and newnode

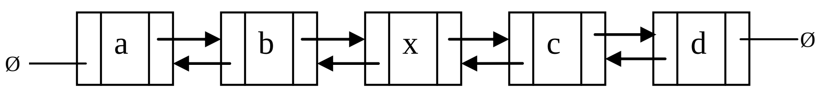
```
typedef char ListEntry;
typedef struct listnode {
  ListEntry entry;
  struct listnode *next;
  struct listnode *previous;
} ListNode;

typedef struct list {
  int count;
  ListNode *current;
  int currentPosition;
} List;
List *list;
```

Original List



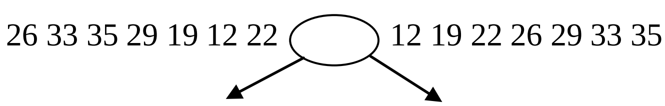
Modified List



4. (8 marks) The root of the recursion tree for Mergesort of the 7 integers **26 33 35 29 19 12 22** is given below. Complete the recursion tree; for each node in the recursion tree, place a list to the left of the node and to the right of the node as follows:

list to the left of the node is the list **passed to** this call of Mergesort
list to the right of the node is the list **upon the return** of this call to Mergesort.

An example of lists to the left and right of a node is given in the root node below. Assume that integer division is used to divide the list in half (i.e., 7/2=3).



5. (8 marks) With a chained hash table we go directly to one of the linked lists before doing any probes. Explain why the average number of probes for an unsuccessful search is $\lambda = n/t$.
Explain in detail and show all your work.

6. (4 marks) Suppose we wish to insert a key into a hash table, but the first probe results in a collision at hash address h. In the following, fill in the next 2 hash addresses that are probed, assuming all probes result in collision:

i) for quadratic probing:
   first probe: h
   next probe:
   next probe:

ii) for linear probing:
   first probe: h
   next probe:
   next probe:

7. (8 marks) Insert the keys **A,Z,B,Y,C,X**, in the order shown, to build them into an AVL tree.
For each insertion:
a) Draw the tree before any necessary rotations. If no rotations are required for this insertion, skip (b) and (c)
b) Write what kind of rotation is necessary, if any, (e.g., "double-left")
c) Draw the final tree after the rotation (you may show intermediate work here if desired.)

8. (8 marks) Complete the function TreeSearch, which searches for a target in a binary search tree. Assume the given declarations, Pre-conditions, and Post-conditions.

```
typedef struct treenode TreeNode;
typedef struct treenode {
  TreeEntry entry;
  TreeNode *left;
  TreeNode *right;
} TreeNode;
```

/*Pre: The tree to which root points has been created.
  Post: The function returns a pointer to a node that matches target, or NULL if the target is not in the tree
*/

TreeNode *TreeSearch(TreeNode *root, KeyType target) {