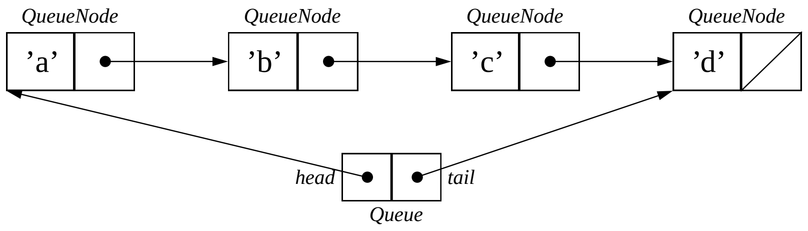
McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 2S03
Exercise Sheet 11

SFWR ENG 2S03 — Principles of Programming

24 November 2006

Exercise 11.1 — Queues



Let the queue datatypes of the lecture be given:

```
typedef struct QueueNodeStruct {
      char data;
      struct QueueNodeStruct * nextPtr;  /* struct label necessary! */
   } QueueNode;

typedef struct {
      QueueNode * head, * tail;
   } Queue;
```

Implement $O(1)$ concatenation of queues.

Exercise 11.2 — Doubly-Linked Lists

(a) Provide type definitions for doubly-linked lists with tail pointers. (These are essentially like queues, except that there are also backward-links between the list nodes.)

(b) Implement $O(1)$ adding of a single element at the head.

(c) Implement $O(1)$ adding of a single element at the tail.

(d) Implement $O(1)$ deletion of the first element.

(e) Implement $O(1)$ deletion of the last element.

(f) Implement $O(1)$ concatenation.

(g) Implement ordered insertion, i.e., insertion into an ordered list such that the ordering is preserved.

(h) Write a *main* function **in a separate module** to test all the above.

A **binary tree** with *int* data is
– either the empty tree (NULL)
– or (pointer to) a node with an *int* data field and two successor trees.

```
typedef struct TreeNodeStruct {
    struct TreeNodeStruct *leftPtr;   // (pointer to) left subtree
    int data;                         // node value
    struct TreeNodeStruct *rightPtr;  // (pointer to) right subtree
} TreeNode;

typedef TreeNode * Tree;
```

Exercise 11.3 — Printing of Binary Trees

Write a recursive function *printTree* to print a binary tree to the screen. The function should print the tree row by row with the top of the tree at the left of the screen, and each level of the tree indented 5 spaces. Since turning your head to the left by 90 degrees should give you the "normal" picture of the tree, for each node, the right subtree has to be printed first, then the node itself, then the left subtree.

(For more explanation, see Exercise 12.25 in the textbook. **Note:** The algorithm proposed there is not purely recursive!)

Exercise 11.4 — Searching in Binary Trees

Write a function *treeSearchNode* that takes a tree and a value as arguments and returns NULL if the value is not in the tree, and otherwise returns a pointer to the node containing the value.

(Exercise 12.23 from the textbook)

Exercise 11.5 — Deleting from Binary Trees

Use the function *cutOffLargestNode* from the lecture to implement a function for deleting a value from an ordered binary tree such that the result ot the deletion is again an ordered binary tree.

(For more explanation, see Exercise 12.22 in the textbook.

Exercise 11.6 — Breadth-First Traversal of Binary Trees

Do exercise 12.24 from the textbook.

Then, make a **second** version of this program and, in that second version, replace the queue with a stack: replace *enqueue* with *push* and *dequeue* with *pop*. What does the program do now?

Exercise 11.7 — Test for Ordered Trees

A binary tree is **ordered** if and only if for every node *n*,
– every value in the left subtree of *n* is not greater than the value in *n*, and
– every value in the right subtree of *n* is not less than the value in *n*.

Design and implement a C function `bool isOrdered( Tree t )`
that returns true if its argument tree is ordered, and false otherwise.