EE 2331 Data Structures and Algorithms, Semester B, 2009/10

Tutorial 3: Linked Lists

Week 3 (4th February, 2010)

The tasks of tutorial exercises are divided into three levels. Level 1 is the basic tasks. You should have enough knowledge to complete them after attending the lecture. Level 2 is the advanced tasks. You should able to tackle them after revision. Level 3 is the challenge tasks which may be out of the syllabus and is optional to answer. I expect you to complete at least task A in the tutorial.

Outcomes of this tutorial

1. Able to traverse a singly linked list
2. Able to insert a node into a singly linked list
3. Able to delete a node from a singly linked list
4. Able to understand pass-by-reference and pass-by-value

Linked lists are a widely used data structure for storing a dynamic number of items. Unlike an array, whose maximum capacity is determined on its creation, a linked list can accommodate any number of data items provided that memory can be allocated. In this tutorial, we will use a singly linked list with dummy header to store and operate on a number of integers.

A singly and linear linked list with dummy header can be visualized as below:

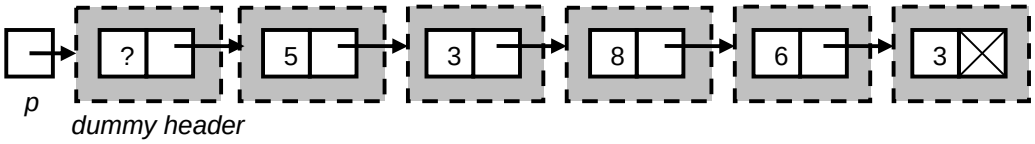

Fig. 1 An example linked list with dummy header pointed by *p*.

In C, each node can be represented as a C structure, and the list can be handled by a pointer to a dummy header node and then the first node of the list. An empty list is represented by a dummy header node followed by a NULL pointer.

The integers are stored in a text file with the following format:

Row	Content	Remark
1st	*n*	The number of integers in this file
2nd	$a_1\ a_2\ a_3 \ldots a_n$	The *n* integers

The routine of reading the integers from text file will be provided to you. You should concentrate on completing the following tasks only:

Task A (Level 1): Search

Write a C function, **Node *search(Node *p, int x, int *count)**, that accepts a pointer *p* to a list of integers and an integer *x* and returns a pointer to the first node containing *x*, if it exists, and the null pointer otherwise. The function also sets variable *count* to the value equals the number elements containing *x* in the linked list. You should not modify the linked list structure and the value of each node.

For example, *p* is the linked list in figure 1:

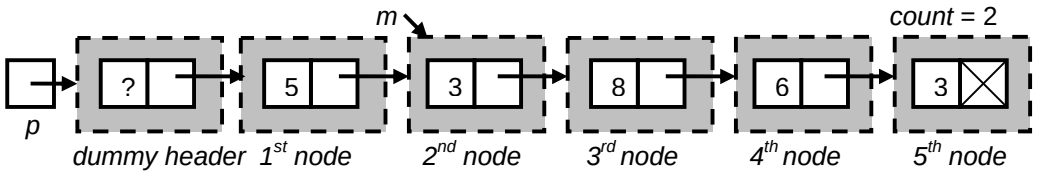

Fig. 2 The linked list after calling search(*p*, 3, &count). It returns a pointer (*m*) which points to the first node containing 3. It also sets variable count to 2 since two nodes in the list have been found containing value 3.

Expected Output:

```
Enter the file name for testing: test1.txt
The linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
Please enter your action ( 1) search, 2) insert, 3) delete ): 1
Please enter the value of x: 3

The searched node is: 3
The number of nodes containing x is: 2
The final linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
```

```
Enter the file name for testing: test1.txt
The linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
Please enter your action ( 1) search, 2) insert, 3) delete ): 1
Please enter the value of x: 4

The searched node is: not found
The number of nodes containing x is: 0
The final linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
```

Discussion: How can you return the pointer of the first searched node? Why do we pass *&count* to the function instead of *count*?

Task B (Level 2): Insert

Write another function, **void insert(Node *_p_, int _x_)**, that adds *x* to the end of *p* if *x* does not exist in the linked list.

For example, *p* is the linked list in figure 1:

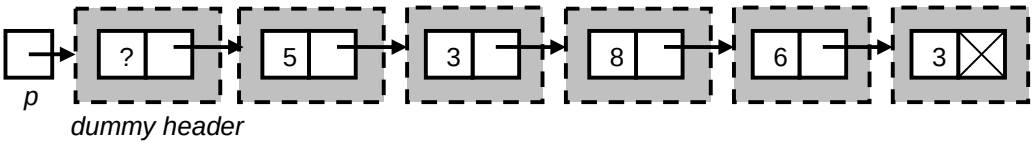

Fig. 3 The linked list after calling insert(*p*, 3). Since the linked list contains 3 already, no nodes would be inserted.

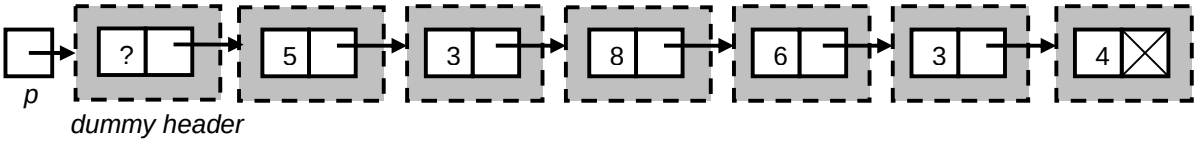

Fig. 4 The linked list after calling insert(*p*, 4). Since the linked list does not contain 4, a new node has been inserted.

Expected Output:

```
Enter the file name for testing: test1.txt
The linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
Please enter your action ( 1) search, 2) insert, 3) delete ): 2
Please enter the value of x: 3

The final linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
```

```
Enter the file name for testing: test1.txt
The linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
Please enter your action ( 1) search, 2) insert, 3) delete ): 2
Please enter the value of x: 4

The final linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> 4 -> NULL
```

Discussion: Is there any difference between inserting a node at the end of the linked list and inserting in the middle of the linked list?

Task C (Level 2): Delete

Write the third function, **int delete(Node *p, int x)**, that deletes the ALL nodes containing *x* from *p*, and return the number of nodes that have been deleted.

For example, *p* is the linked list in figure 1:

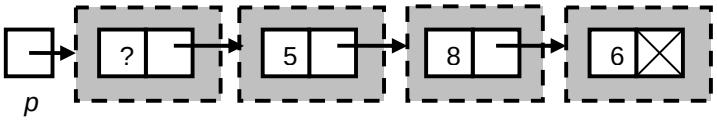

Fig. 5 The linked list after calling delete(*p*, 3). The function returns 2 indicating two nodes have been deleted.

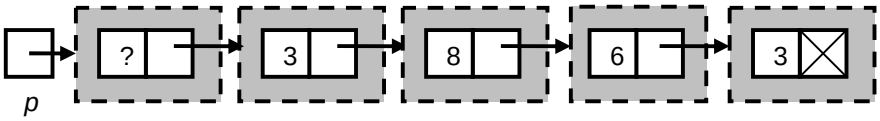

Fig. 6 The linked list after calling delete(*p*, 5). The function returns 1 indicating one node has been deleted.

Expected Output:

```
Enter the file name for testing: test1.txt
The linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
Please enter your action ( 1) search, 2) insert, 3) delete ): 3
Please enter the value of x: 3

The number of nodes deleted is: 2
The final linked list is: 5 -> 8 -> 6 -> NULL
```

```
Enter the file name for testing: test1.txt
The linked list is: 5 -> 3 -> 8 -> 6 -> 3 -> NULL
Please enter your action ( 1) search, 2) insert, 3) delete ): 3
Please enter the value of x: 5

The number of nodes deleted is: 1
The final linked list is: 3 -> 8 -> 6 -> 3 -> NULL
```

Task D (Level 3): Pass-by-value and pass-by-reference of pointers

Comment the function **void printout(Node *p)** whether the argument *p* is pass-by-value or pass-by-reference. How to justify your answer? Please demonstrate your work to tutors.