COMP2401—Tutorial 7
linked-lists and valgrind tool

Learning Objectives
After this tutorial, you will be able to:
- Build basic linked-list operations
- Check if your program has a memory leak using the valgrind tool

Submit your tutorial as t7.tar

Tutorial
Download the file myLinkedList.c

1 Linked List Creation

1.1 Introduction
In this part you will implement several basic linked list functions. The linked list is designed to manipulate and store employee records. Here we term the field that holds the employee information "data". The reason is that in general one would implement a generic linked list that can be used with any data type (using void *).

Get **myLinkedList.c** in which you are given employee structures and operations as well as the linked list structures and the main function. You have to complete the following six functions for the linked list:

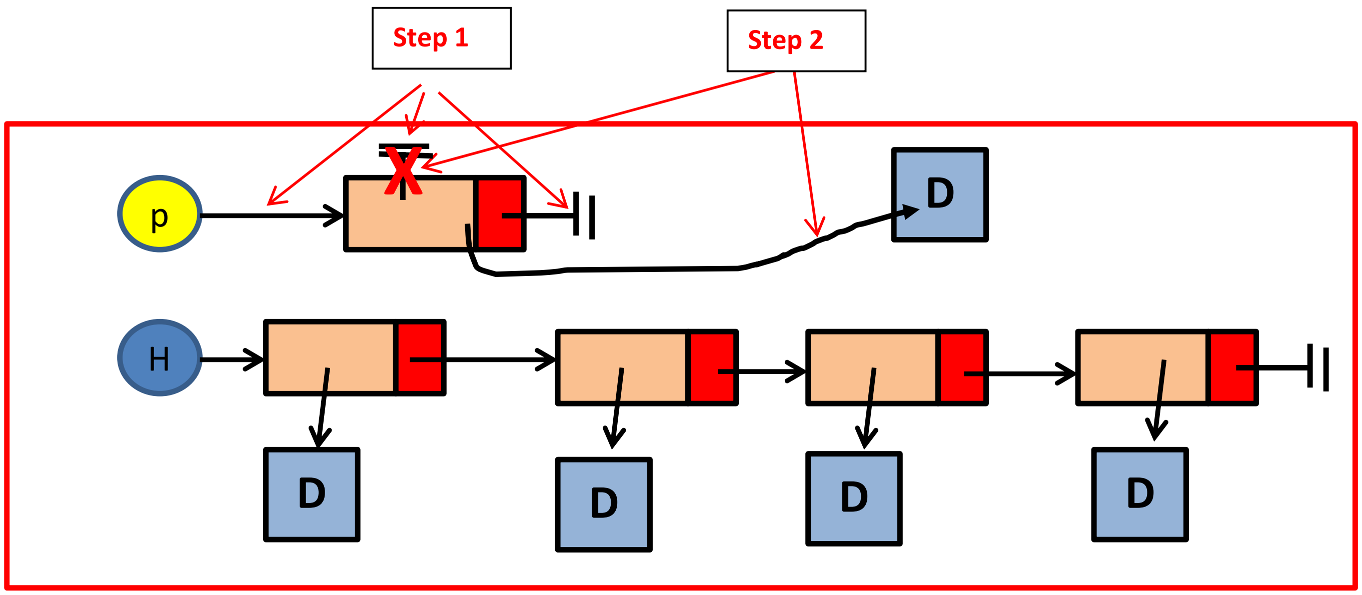
a. **addNode** –adds a record as the first node of the linked list
b. **printList** – print the list from the first node to the last node
c. **deleteNode** – delete the first node from the linked list.

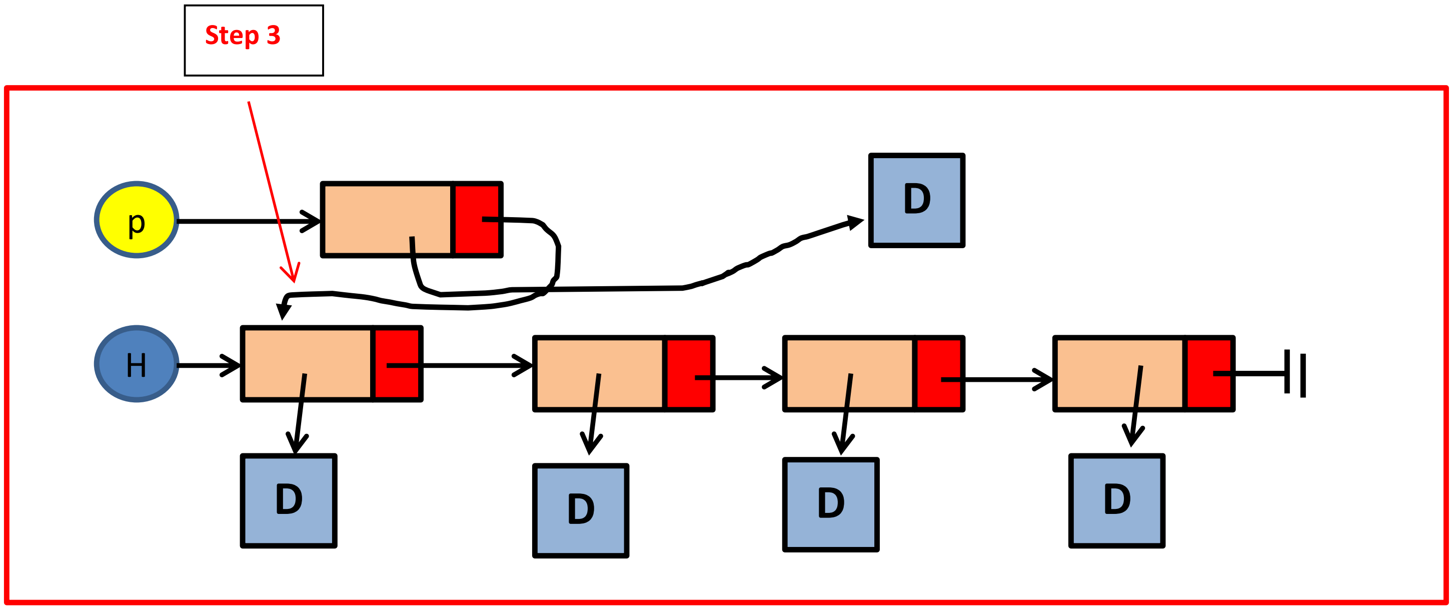
1.2 Task 1
Implement the function addNode(). This function should add the new employee to the linked list as the first node.

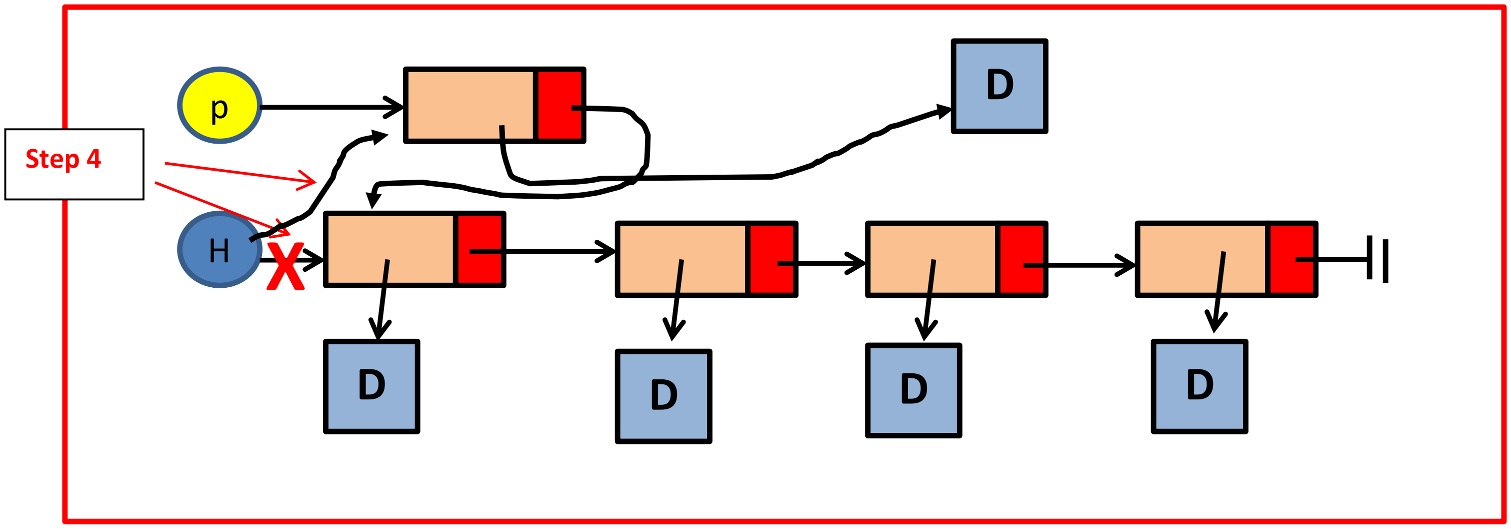
Pseudo code

```
ListNode *p = NULL;

// Step 1 allocate memory for the node and initialize all the pointers to NULL

// Step 2 assign the employee record to data
// Step 3 update the next field of the new node to point to the first node in the
list as a next node

// Step 4 update the list head to point to the new node.
```

Check your code using **valgrind**. Valgrind should show that five blocks of code were used.







1.3 Task 2

Implement the function printList(). This function should print the linked list from the first node to the last node.
This function should be implemented using the iterative approach of traversing the list.

This is a simple "for" loop. The function needs to traverse the list until it reaches the sentinel condition. Here the sentinel is a NULL. The sentinel indicates that end of the list when the pointer "next" is NULL.

Note, that here one can take advantage of the fact that the parameter "head" is a call by value and therefore its value can be changed without any side effects outside the function. Be sure to use the printEmployee function in employee.c

Pseudo code

```
// While list is not empty
        //Print the node using the printEmployee() function
        // advance to the next node
```

1.4 Task 3

Implement the function deleteNode(). This function should delete the first node in the linked list

The function should return the employee record that is stored in the node and then delete it.

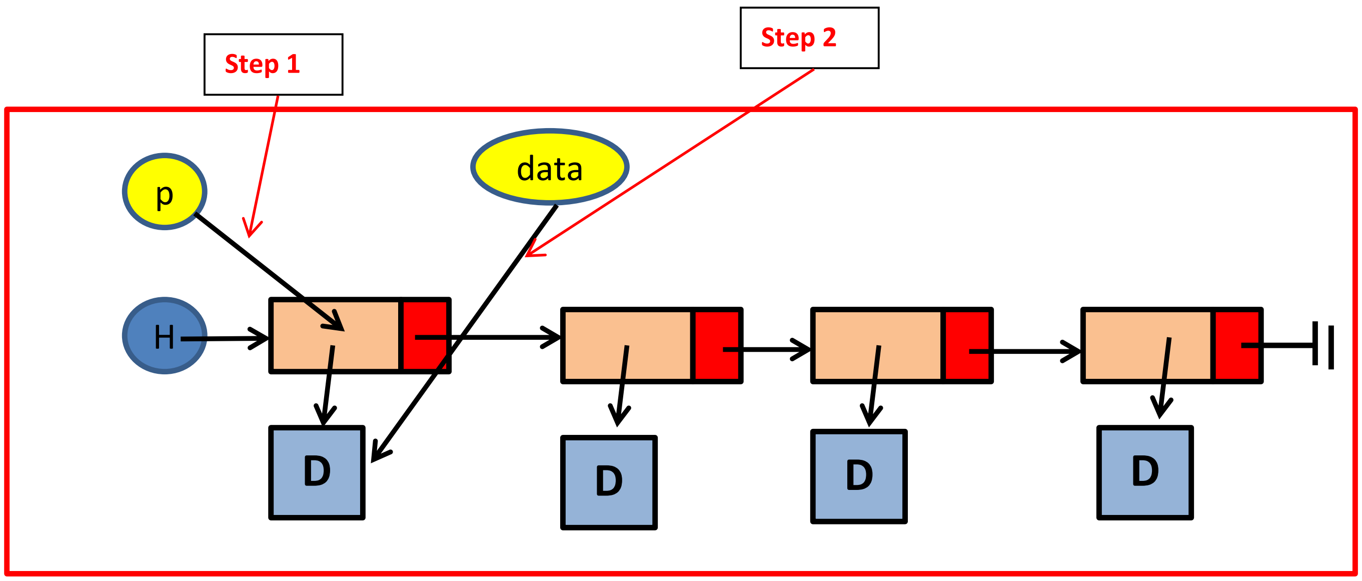
Notes
1. Here head and data are declared as **. This is because each of them is a pointer and the function must create a side effect outside the function.

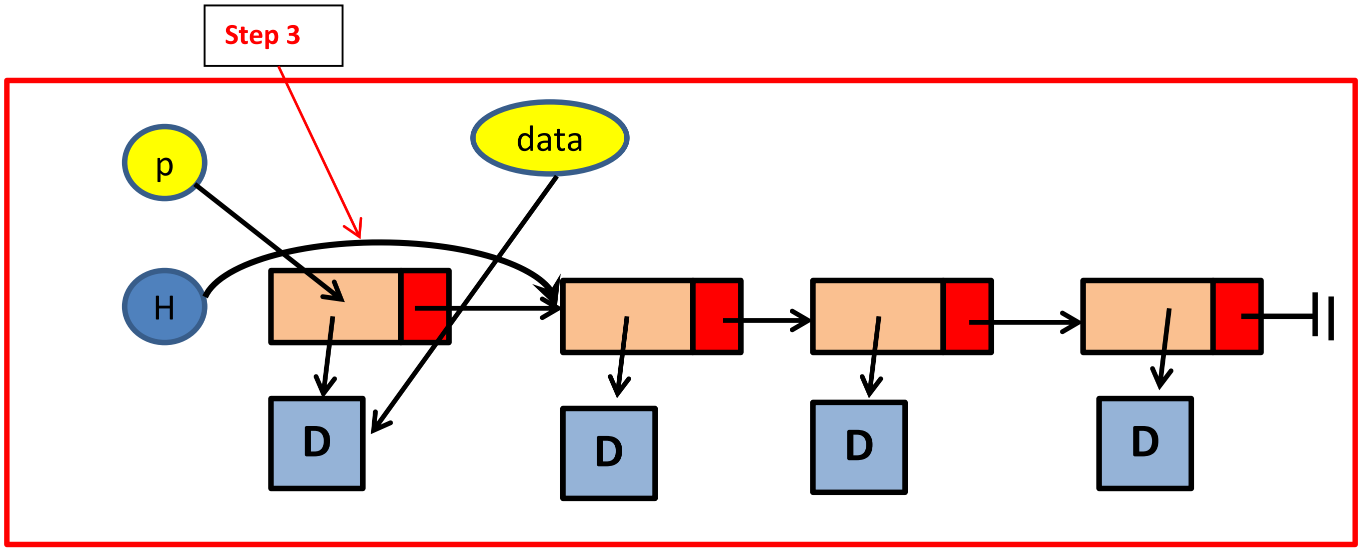
Output – the address of the employee record that was stored in the linked list

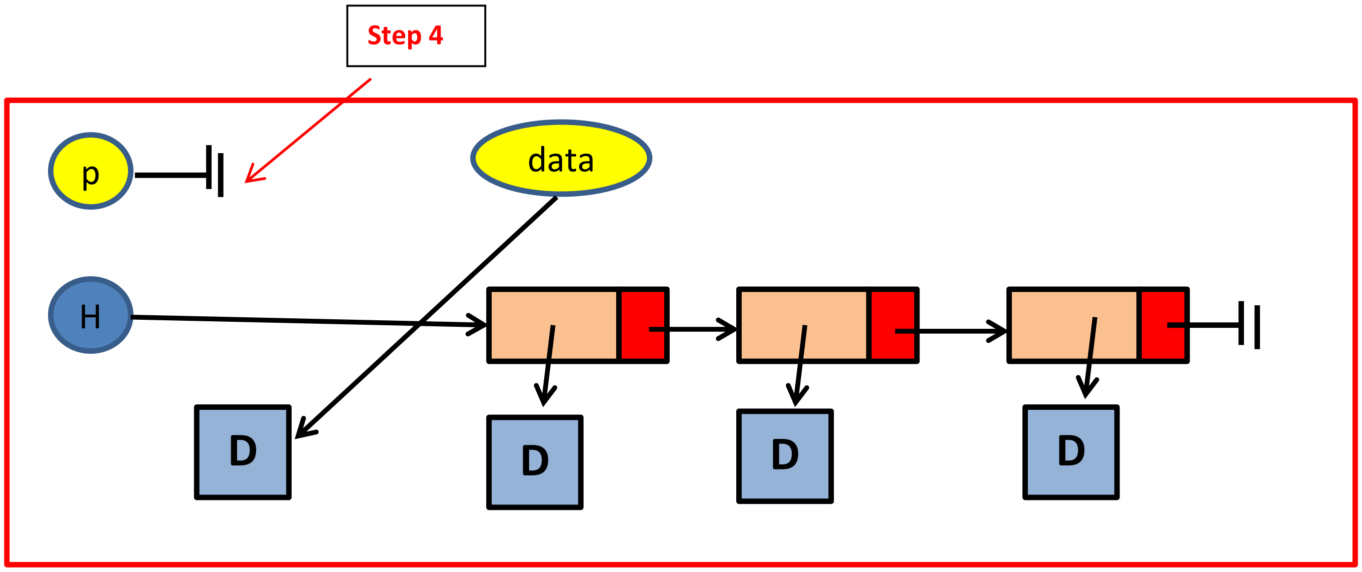
<u>Pseudo code</u>

```
ListNode *p = NULL;

// if the linked list is empty then
      // set output data to NULL
      // return

// Step 1 assign to p the address of the first node

// Step 2 assign data in node to the output data

// Step 3 update the head to point to the next node in the list (head->next)
      // this can be done either by using p as *head = p->next;
      // or by using the head as *head = *head->next

      Can you recall why one needs to use (*head)->next?

// Step 4 free the node pointed to by p and set p to NULL
```

Check your code using **valgrind**. Valgrind should show that no blocks of memory were lost.







Submit your tutorial work (1 file) in a tar file t7.tar!