Analysis & Design of Algorithms

Semester III 2018-19

Lab - 2

Topics: Data Structure : Linked List, Stacks

LINKED LIST

After getting familiar with creating simple C programs, I/O operations and pointers in C, let us get started with our first data structure for this course âĂŞ âĂIJLinked ListsâĂİ. We will use all of the above we have learnt in this exercise.

Linked List Basics

A linked-list is a sequence of data structures which are connected together via links. Linked List is a sequence of nodes which contains items. Each node contains a connection to another node via a link. Following are important terms to understand the concepts of Linked List.

- Node - Each node of a linked list can store data called an element.
- Next - Each node of a linked list contains a link to the next node called "next". This is typically a pointer variable of type node.
- LinkedList - A LinkedList or head of the linked list contains the connection link to the first node, called "first".

As per above shown illustration, following are the important points to be considered.

- LinkedList / head contains a link element called "first", which points to the first node in the list.
- Each node carries a data field and a Link Field called next.
- Each node is linked with its next node using its next link.
- Last node carries a Link as null to mark the end of the list. Types of Linked List

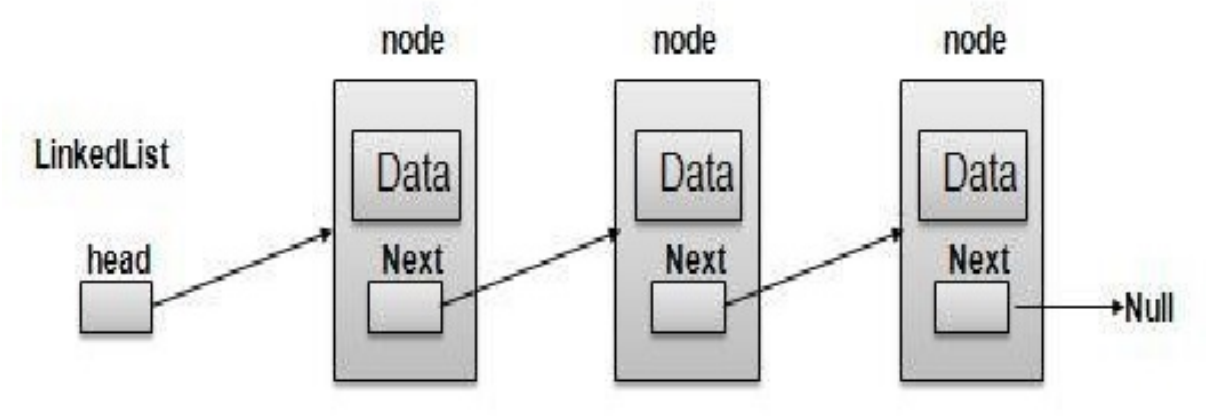

Figure 1: Linked List

Following are the various flavours of linked list.

- Simple Linked List - Item Navigation is forward only.
- Doubly Linked List - Items can be navigated forward and backward way.
- Circular Linked List - Last node contains link of the first element as next and first element has link to last element as prev.

In this exercise we will work on Simple Linked List that store integers as data elements in their nodes. Following are the basic operations supported by it.

- Insertion - add an element at the beginning of the list.
- Deletion - delete an element at the beginning of the list.
- Display - displaying complete list.
- Search - search an element using given key.
- Delete - delete an element using given key.

```
#include <stdio.h>
// structure definitions
// structure of a linked list node. It contains an element
struct node {
  int element;
  struct node * next; };
```

```c
/* structure of a linked list / head. It stores the count of number of elements in
    the list and also a pointer link to the first node of the list. */
struct linkedList {
  int count;
  struct node * first; };
// function declarations
void insertFirst (struct linkedList * head, int ele);  /* inserts a given element at
    the beginning of the list */
struct node * deleteFirst(struct linkedList * head);
/* deletes the first element of the list and returns pointer to the deleted node. */
void printList (struct linkedList * head);
/* prints all the elements in the list */
int search (struct linkedList * head, int ele);
/* searches for a given element in the linked list. Returns 1 if found, 0 otherwise.
    */
struct node * delete (struct linkedList * head, int ele);
/* deletes the first node in the list that contains the element = ele and retuns it.
    If the element is not found it returns an error message saying element not found.
    */
```

Now let us study each of the above operations we have declared and implement them

Insertion Operation (Inserts an element at the beginning of the list)

Insertion is a three step process -

- Create a new node with provided data.
- Point New node to old First node.
- Point "First" link in the head to this New node.
- Increment the count in the head.

```c
//insert link at the first location
void insertFirst(struct linkedList * head, int ele){
  //create a node
  struct node *link = (struct node*) malloc (sizeof(struct node)); /* by this you are
     creating a node whose address is being stored in the link pointer. */
  link->element = ele;

  //point it to old first node
  link->next = head->first;
```

```
9
10   //point first to new first node
11   head -> first = link;
12   head -> count --;
13 }
```

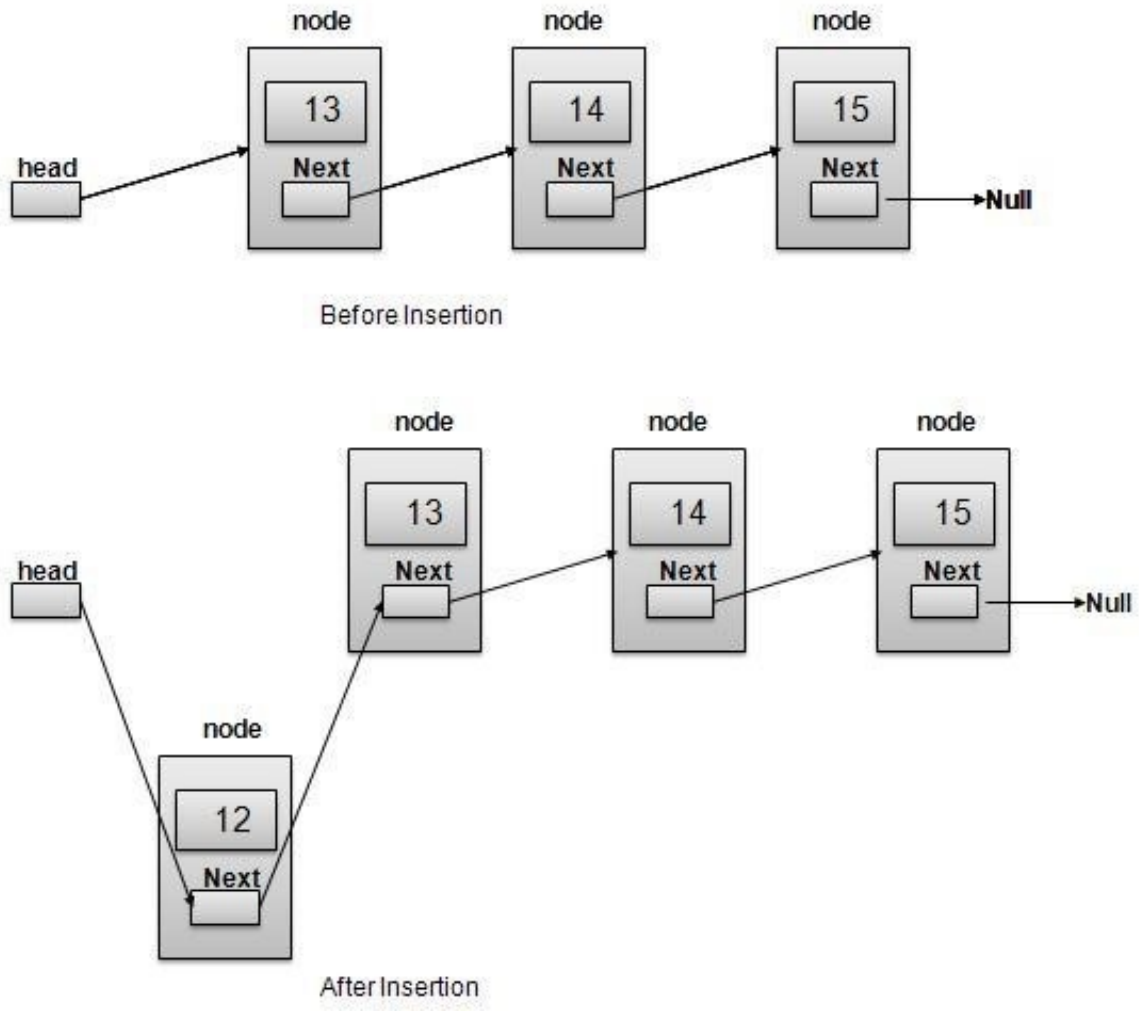

Figure 2: Insertion of node

Deletion Operation

Deletion is a two step process

- Get the Link pointed by First Link as Temp Link.
- Point First Link to Temp Link's Next Link.

- Decrement the count.

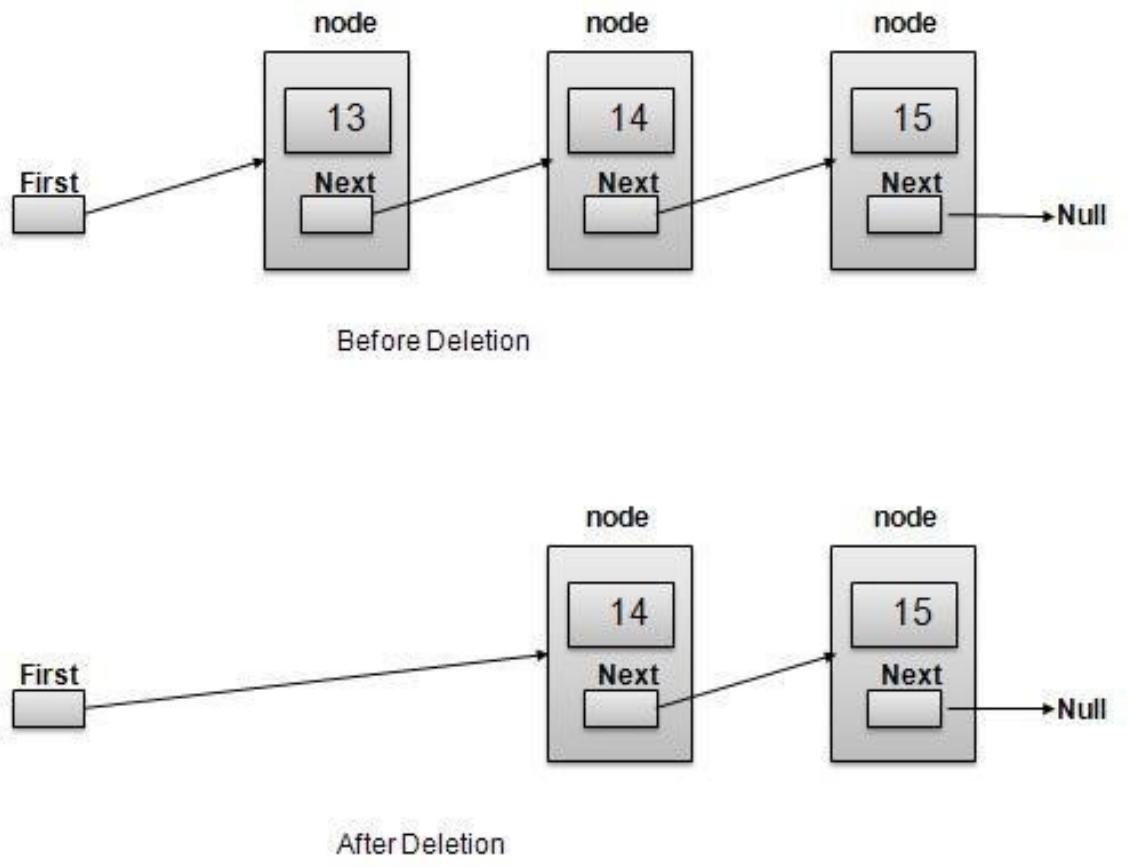

Figure 3: Insertion of node

```
//delete first item
struct node* deleteFirst(struct linkedList * head){
  // exercise to implement this operation.
}
```

Printing the list (Navigation)

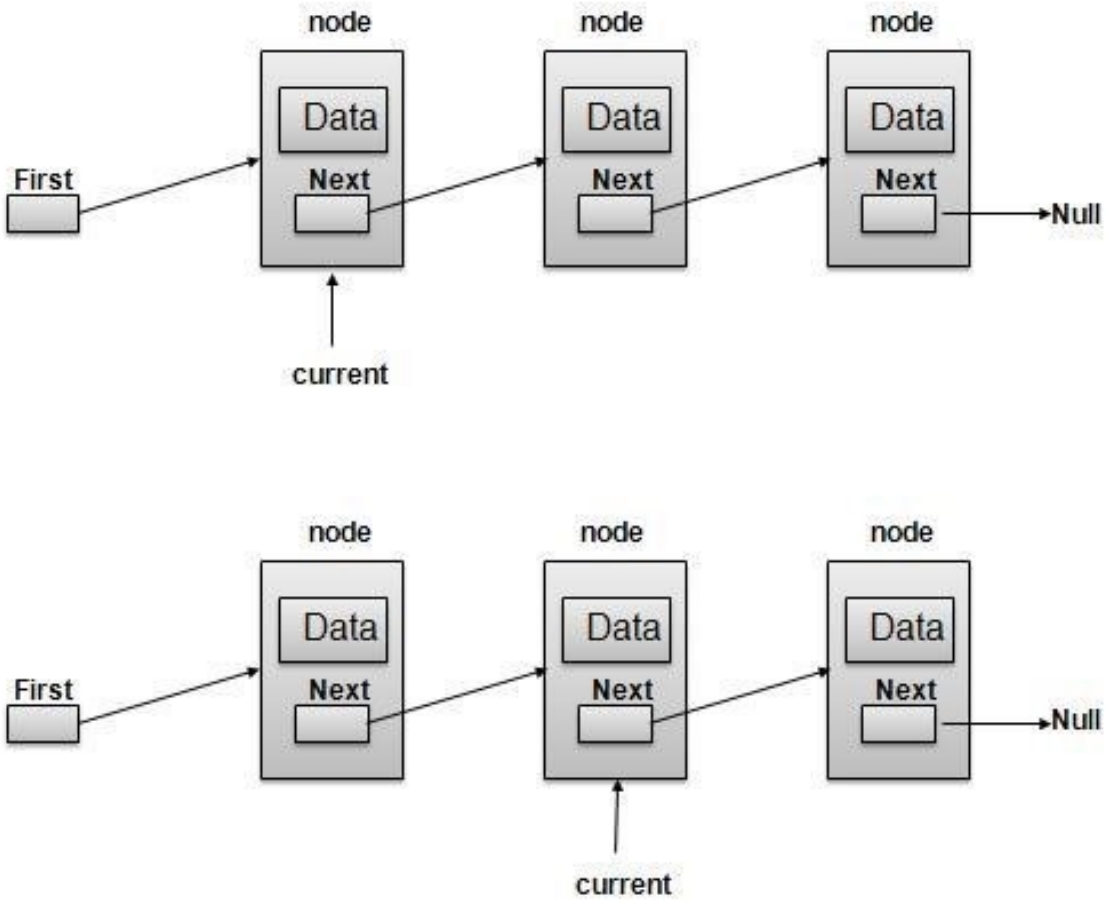

Figure 4: Printing the list

Navigation is a recursive step process and is basis of many operations like search, delete etc.

- Get the node pointed by First Link as Current link.
- Check if Current link is not null and display it.
- Point Current link to the Next of Current Link and move to above step.

```
//display the list
void printList(struct linkedlist * head){
struct node *ptr = head->first;
printf("\n[ ");
```

```
//start from the beginning
while(ptr != NULL){
  printf("%d, ", ptr->element);
  ptr = ptr->next;
  }
printf(" ]"); }
```

The main file:

```
#include <stdio.h> #include âĂIJlinkedlist.hâĂİ
int main(int argc, char *argv[])
{
  // Create a file pointer and open the file read as command line argument.
  FILE * fileptr = fopen(argv[1], r);
  // Declare a pointer to a linked list (head) and allocate memory to it.
  struct linkedList * head = (struct linkedList *) malloc (sizeof(struct linkedList))
    ;
  // In a loop read the file and insert elements into the linkedList.
  while (!feof(fileptr))
  {
    // read the next element and store into the temp variable.    int temp;
    fscanf(fileptr, âĂIJ%d âĂİ, &temp);

    // insert temp into the linked list.
    insertFirst(head,temp);

  }
  // close the file pointer   fclose(fileptr);
  // print the linked list.
  // delete the first element of the linked list.
  // print the linked list again to check if delete was successful.
  // print the linked list to a new file.
}

Create sampleInput.txt to contain data as follows:
32
49
45
9
14
23
17
95
```

Exercise

3.a: Write a C program that reads the contents of a file (containing a sequence of characters or a string) into a linkedList. Perform insertion and deletion operation. Find out whether the character string stored in array is palindrome or not. To accomplish the task, copy the content of linkedlist into another array and check for palindrome by comparing content of each element in both the linkedlist. Pass the filenames as command line arguments.

3.b: Write a C program that reads the content of two different files in two separate linkedlist. Both the files contain sequence of integer numbers separated by space in ascending order. Write a function which will traverse these two linkedlist and merge them without disturbing the order. Merged list should be then copied to an output file. Pass file names as command line arguments.

3.c Implement Last-In-First-Out (Stack) data structure using the Linked List created above. Stack supports two operations: Push and Pop. Push inserts a given element at the front of the stack. And Pop deletes the first element and returns it. You can create stackInput.txt

Optional only for Using Linux

Create following files in your directory

1. **driver.c** : contains the main program
2. **linkedlist.c** or any name.c : which contains all the implemented function. Shall contain all the necessary operations for insertions, deletions and search operations in a linked list.
3. **name.h** : which shall contain all basic definitions of the structures and functions associated with operations of linked list. eg: linkedlist.h

Definition of linkedlist.h

```
1 /* linkedlist.h */
2 #include <stdio.h>
3 // structure definitions
4 struct node {
5 int element;
6 struct node * next; };
```

```
7
8  struct linkedList {
9  int count;
10 struct node * first; };
11
12 // function declarations
13 void insertFirst (struct linkedList * head, int ele);
14 struct node * deleteFirst(struct linkedList * head);
15 void printList (struct linkedList * head);
16 int search (struct linkedList * head, int ele);
17 struct node * delete (struct linkedList * head, int ele);
```

Compiling the C programs and files

Now let us try to compile, link and execute the above code. We have two .c files âĂŞ linkedlist.c and driver.c. We need to compile them separately as follows:

$ gcc -c linkedlist.c

$ gcc -c driver.c

This would create two files - linkedlist.o and driver.o. You need to link them into a single executable file as follows and then run it with sampleInput.txt as the command line argument. The program would read the input required from this file.

$ gcc -o exe linkedlist.o driver.o

$ ./exe sampleInput.txt