DATA STRUCTURE - LINKED LIST

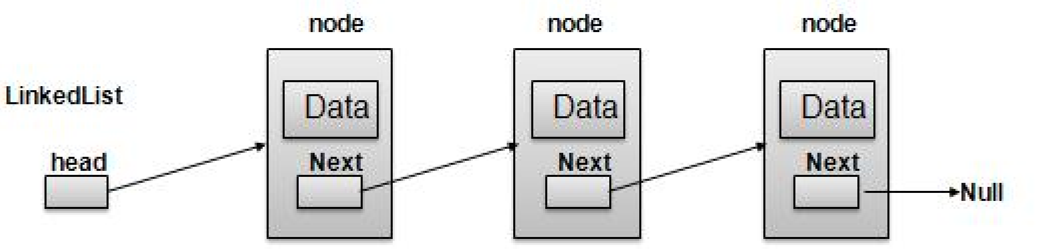
Linked List Basics

A linked-list is a sequence of data structures which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list the second most used data structure after array. Following are important terms to understand the concepts of Linked List.

- **Link** − Each Link of a linked list can store a data called an element.
- **Next** − Each Link of a linked list contain a link to next link called Next.
- **LinkedList** − A LinkedList contains the connection link to the first Link called First.

Linked List Representation



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

- LinkedList contains an link element called first.
- Each Link carries a data fields and a Link Field called next.
- Each Link is linked with its next link using its next link.
- Last Link carries a Link as null to mark the end of the list.

Types of 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 item contains link of the first element as next and and first element has link to last element as prev.

Basic Operations

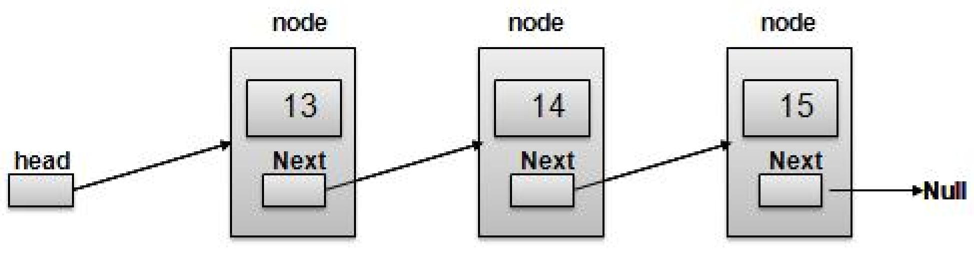
Following are the basic operations supported by a list.

- **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.

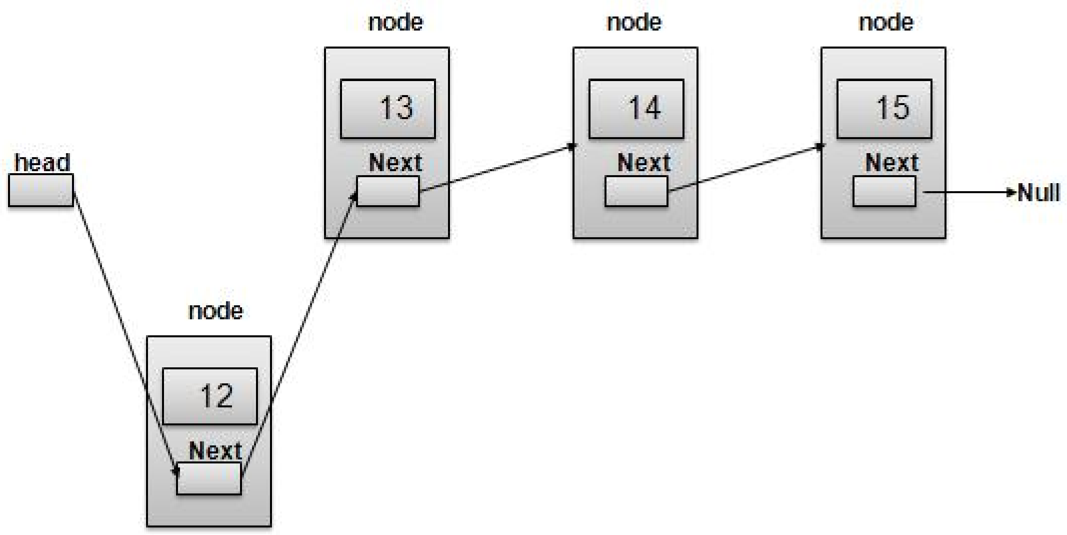
Insertion Operation

Insertion is a three step process −

- Create a new Link with provided data.
- Point New Link to old First Link.
- Point First Link to this New Link.



Before Insertion



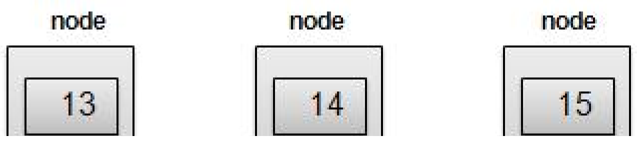
After Insertion

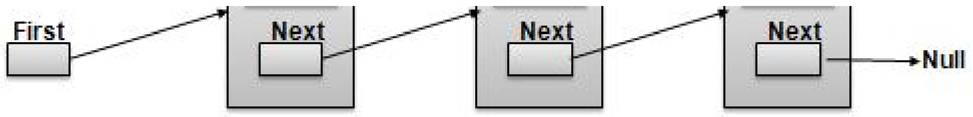
```
//insert link at the first location
void insertFirst(int key, int data){
   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

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

   //point first to new first node
   head = link;
}
```

Deletion Operation

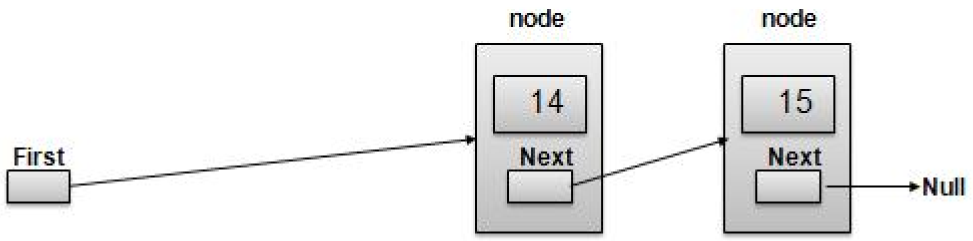
Deletion is a two step process −

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





Before Deletion



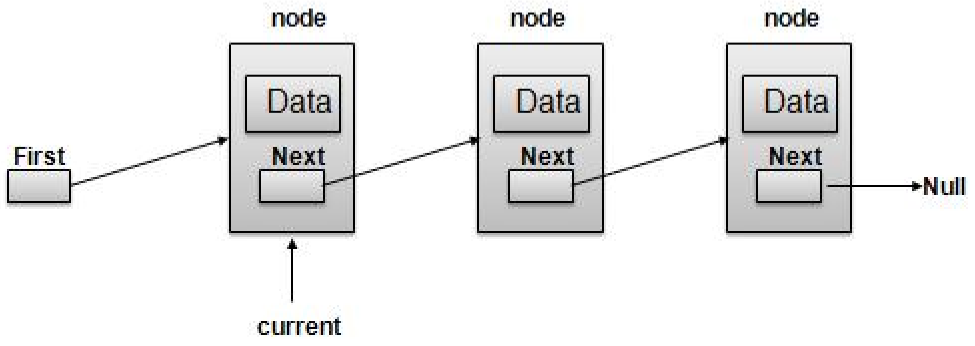
After Deletion

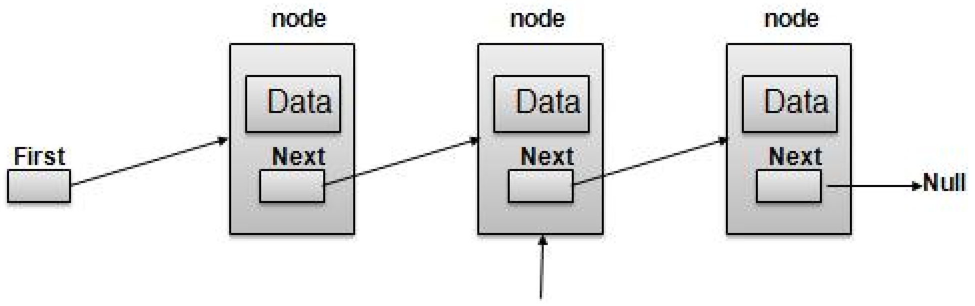
```
//delete first item
struct node* deleteFirst(){
   //save reference to first link
   struct node *tempLink = head;

   //mark next to first link as first
   head = head->next;

   //return the deleted link
   return tempLink;
}
```

Navigation Operation

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

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





current

Note −

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

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

   printf(" ]");
}
```

Advanced Operations

Following are the advanced operations specified for a list.

- **Sort** − sorting a list based on a particular order.
- **Reverse** − reversing a linked list.

Sort Operation

We've used bubble sort to sort a list.

```
void sort(){

   int i, j, k, tempKey, tempData ;
   struct node *current;
   struct node *next;
   int size = length();
   k = size ;

   for ( i = 0 ; i < size - 1 ; i++, k-- ) {
      current = head ;
      next = head->next ;

      for ( j = 1 ; j < k ; j++ ) {

         if ( current->data > next->data ) {
            tempData = current->data ;
            current->data = next->data;
            next->data = tempData ;

            tempKey = current->key;
            current->key = next->key;
            next->key = tempKey;
         }

         current = current->next;
         next = next->next;
      }
   }
}
```

Reverse Operation

Following code demonstrate reversing a single linked list.

```
void reverse(struct node** head_ref) {
   struct node* prev   = NULL;
```

```
    struct node* current = *head_ref;
    struct node* next;

    while (current != NULL) {
       next  = current->next;
       current->next = prev;
       prev = current;
       current = next;
    }
    *head_ref = prev;
}
```

To see linked-list implementation in C programming language, please click here.