Chapter - 5

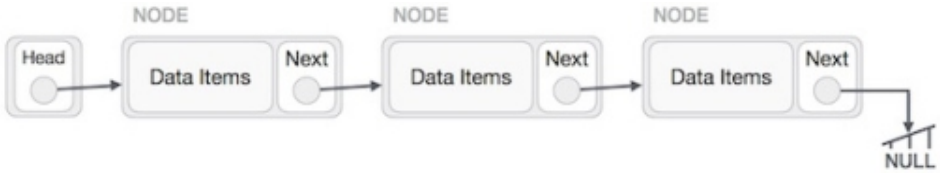
Linked List:

Linked list is a linear data structure that contains sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called as "Node".

Simply a list is a sequence of data, and linked list is a sequence of data linked with each other. linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

Node – Each node contains data item and a pointer which is address of next node in list
Next – A pointer field which contains address of next node in list

Linked List Representation:
Linked list can be visualized as a chain of nodes, where every node points to the next node.



Type
Follo
Single Linked List – Item navigation is forward only.
Doubly Linked List – Items can be navigated forward and backward.
Circular Linked List – Last item contains link of the first element as next and the first element has a link to the last element as previous.

Advantages of Linked list: Following are advantages of linked list-
(a) Linked List is Dynamic Data Structure.
(b) Linked List can grow and shrink during run time.
(c)Insertion and Deletion Operations are easier
(d) Efficient Memory Utilization, i.e, no need to pre-allocate memory
(e) Faster Access time can be expanded in constant time without memory overhead
(f) Linear Data Structures such as Stack, Queue can be easily implemented using Linked list

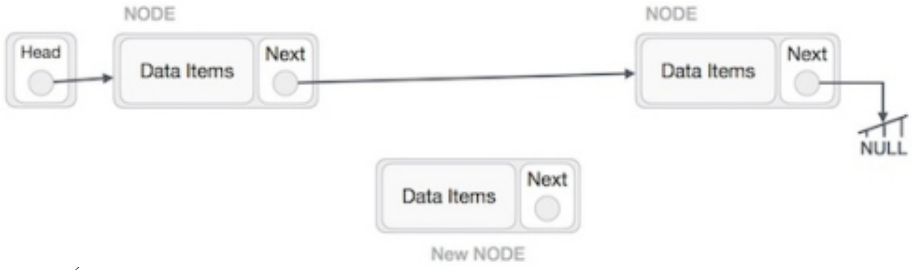
Disadvantages of Linked list:
(a) Memory wastage if required space is known
(b) Searching operations is difficult.

Basic Operations:
Basic operations supported by a list are
Insertion – Adds an element at the beginning of the list.
Deletion – Deletes an element at the beginning of the list.
Display – Displays the complete list.

Search – Searches an element using the given key.

Insertion Operation:

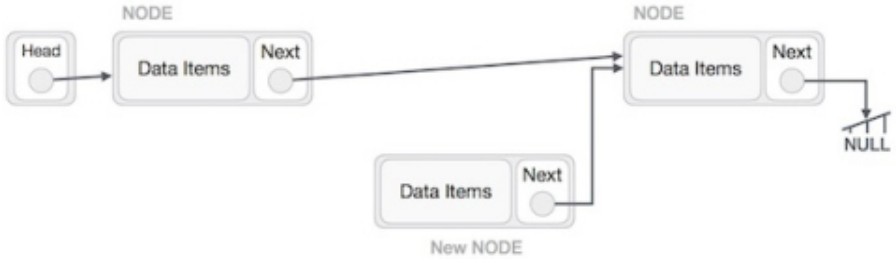
Adding a new node in linked list is a more than one step activity. We shall learn this with diagrams here. First, create a node using the same structure and find the location where it has to be inserted.



Imag ıd C
(Rigl

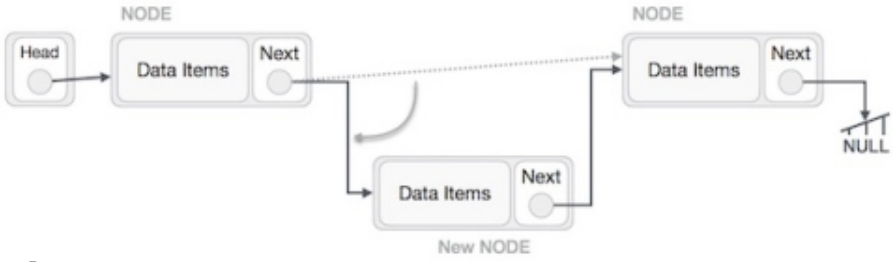
point B.next to C and NewNode.next –> RightNode;

It should look like this –

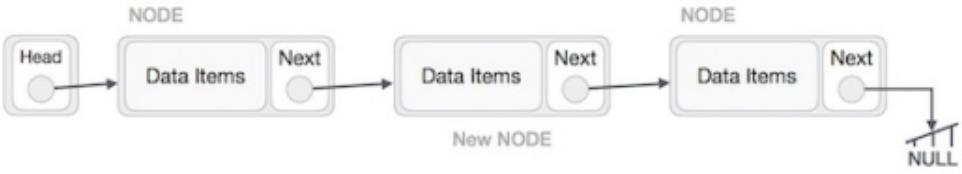


Now,

LeftNode.next –> NewNode;

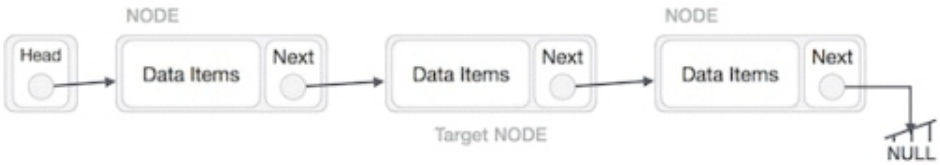


This w



Sin...ile inserting it at the end, the second last node of the list should point to the new node and the new node will point to NULL.

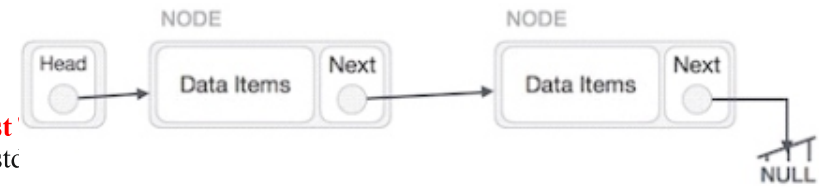
Deletion Operation:
Deletion is also a more than one step process. We shall learn with pictorial representation. First, locate the target node to be removed, by using searching algorithms.



The ...get node –
LeftNode.next –> TargetNode.next;

This ...de, we w...
TargetNode.next –> NULL;

We ...y deallocate memory and wipe off the target node completely.



Linked List
#include <stc

```
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>

struct node {
                int data;
                int key;
                struct node *next;
              };

struct node *head = NULL;
struct node *current = NULL;

//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(" ]");
}

//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;
}

//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;
}

//is list empty
bool isEmpty() {
   return head == NULL;
}

int length() {
   int length = 0;
   struct node *current;

   for(current = head; current != NULL; current = current->next) {
      length++;
   }

   return length;
}

//find a link with given key
struct node* find(int key) {

   //start from the first link
   struct node* current = head;

   //if list is empty
   if(head == NULL) {
      return NULL;
   }

   //navigate through list
   while(current->key != key) {

      //if it is last node
      if(current->next == NULL) {
         return NULL;
      }else {
         //go to next link
         current = current->next;
      }
   }

   //if data found, return the current Link
   return current;
```

```
}

//delete a link with given key
struct node* delete(int key) {

   //start from the first link
   struct node* current = head;
   struct node* previous = NULL;

   //if list is empty
   if(head == NULL) {
      return NULL;
   }

   //navigate through list
   while(current->key != key) {

      //if it is last node
      if(current->next == NULL) {
         return NULL;
      }else {
         //store reference to current link
         previous = current;
         //move to next link
         current = current->next;
      }
   }

   //found a match, update the link
   if(current == head) {
      //change first to point to next link
      head = head->next;
   }else {
      //bypass the current link
      previous->next = current->next;
   }

   return current;
}

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;
      }
   }
}

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;
}

main() {
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);

   printf("Original List: ");
```

```
//print list
printList();

while(!isEmpty()) {
   struct node *temp = deleteFirst();
   printf("\nDeleted value:");
   printf("(%d,%d) ",temp->key,temp->data);
}

printf("\nList after deleting all items: ");
printList();
insertFirst(1,10);
insertFirst(2,20);
insertFirst(3,30);
insertFirst(4,1);
insertFirst(5,40);
insertFirst(6,56);

printf("\nRestored List: ");
printList();
printf("\n");

struct node *foundLink = find(4);

if(foundLink != NULL) {
   printf("Element found: ");
   printf("(%d,%d) ",foundLink->key,foundLink->data);
   printf("\n");
}else {
   printf("Element not found.");
}

delete(4);
printf("List after deleting an item: ");
printList();
printf("\n");
foundLink = find(4);

if(foundLink != NULL) {
   printf("Element found: ");
   printf("(%d,%d) ",foundLink->key,foundLink->data);
   printf("\n");
}else {
   printf("Element not found.");
}

printf("\n");
sort();
```

```
  printf("List after sorting the data: ");
  printList();

  reverse(&head);
  printf("\nList after reversing the data: ");
  printList();
}
```

If we compile and run the above program, it will produce the following result −

```
Original List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10) ]
Deleted value:(6,56)
Deleted value:(5,40)
Deleted value:(4,1)
Deleted value:(3,30)
Deleted value:(2,20)
Deleted value:(1,10)
List after deleting all items:
[ ]
Restored List:
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10) ]
Element found: (4,1)
List after deleting an item:
[ (6,56) (5,40) (3,30) (2,20) (1,10) ]
Element not found.
List after sorting the data:
[ (1,10) (2,20) (3,30) (5,40) (6,56) ]
List after reversing the data:
[ (6,56) (5,40) (3,30) (2,20) (1,10) ]
```

Important Points

- Linked list is a linear data structure that contains sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called as "Node".
- Linear Data Structures such as Stack, Queue can be easily implemented using Linked list.
- Linked List is Dynamic data Structure.

Exercise

Objective type questions.

Q1. Linked lists are best suited
   a. for relatively permanent collections of data
   b. for the size of the structure and the data in the structure are constantly changing
   c. for both of above situation
   d. for none of above situation

Q2. Generally collection of Nodes is called as _____.
   a. Stack
   b. Linked List
   c. Heap
   d. Pointer

Q3. Which of the following is not a type of Linked List
   a. Doubly Linked List
   b. Singly Linked List
   c. Circular Linked List
   d. Hybrid Linked List

Q4. Linked list is generally considered as an example of _____ type of memory allocation.
   a. Static
   b. Dynamic
   c. Compile Time
   d. None of these

Q5. In a circular linked list
   a. Components are all linked together in some sequential manner.
   b. There is no beginning and no end.
   c. Components are arranged hierarchically.
   d. Forward and backward traversal within the list is permitted.

Short answer type questions.

Q1. Define linked list ?
Q2. What is header linked list ?
Q3. Which is better In array and linked list ?
Q4. Define circular linked list ?

Essay type questions.

Q1. Explain doubly linked list ?
Q2. Differentiate between singly and doubly lined list ?
Q3. Which types of memory allocates linked list ?
Q4. Explain the uses of linked list ?

Answers

Ans1. b Ans2. b Ans3. d
Ans4. b Ans5. b

Chapter 6