Linked List Operations

Now that you have got an understanding of the basic concepts behind linked list and their types, its time to dive into the common operations that can be performed.

Two important points to remember:

head points to the first node of the linked list

next pointer of last node is NULL, so if next of current node is NULL, we have reached end of linked list.

In all of the examples, we will assume that the linked list has three nodes 1 --->2 --->3 with node structure as below:

struct node

{

int data;

struct node \*next;

};

How to traverse a linked list

Displaying the contents of a linked list is very simple. We keep moving the temp node to the next one and display its contents.

When temp is NULL, we know that we have reached the end of linked list so we get out of the while loop.

struct node \*temp = head;

printf("\n\nList elements are - \n");

while(temp != NULL)

{

printf("%d --->",temp->data);

temp = temp->next;

}

The output of this program will be:

List elements are -

1 --->2 --->3 --->

How to add elements to linked list

You can add elements to either beginning, middle or end of linked list.

Add to beginning

Allocate memory for new node

Store data

Change next of new node to point to head

Change head to point to recently created node

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = head;

head = newNode;

Add to end

Allocate memory for new node

Store data

Traverse to last node

Change next of last node to recently created node

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

newNode->next = NULL;

struct node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

Add to middle

Allocate memory and store data for new node

Traverse to node just before the required position of new node

Change next pointers to include new node in between

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = 4;

struct node \*temp = head;

for(int i=2; i < position; i++) {

if(temp->next != NULL) {

temp = temp->next;

}

}

newNode->next = temp->next;

temp->next = newNode;

How to delete from a linked list

You can delete either from beginning, end or from a particular position.

Delete from beginning

Point head to the second node

head = head->next;

Delete from end

Traverse to second last element

Change its next pointer to null

struct node\* temp = head;

while(temp->next->next!=NULL){

temp = temp->next;

}

temp->next = NULL;

Delete from middle

Traverse to element before the element to be deleted

Change next pointers to exclude the node from the chain

for(int i=2; i< position; i++) {

if(temp->next!=NULL) {

temp = temp->next;

}

}

temp->next = temp->next->next;

Complete program for linked list operations

Here is the complete program for all the linked list operations we learnt till now. Lots of edge cases have been left out to make the program short.

We suggest you to just have a look at the program and try to implement it yourself.

Also, notice how we pass address of head as struct node \*\*headRef in the functions insertAtFront and deleteFromFront. These two functions change the position of head pointer so we need to pass the address of head pointer and change its value within the function.

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

void display(struct node\* head)

{

struct node \*temp = head;

printf("\n\nList elements are - \n");

while(temp != NULL)

{

printf("%d --->",temp->data);

temp = temp->next;

}

}

void insertAtMiddle(struct node \*head, int position, int value) {

struct node \*temp = head;

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = value;

int i;

for(i=2; inext != NULL) {

temp = temp->next;

}

}

newNode->next = temp->next;

temp->next = newNode;

}

void insertAtFront(struct node\*\* headRef, int value) {

struct node\* head = \*headRef;

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = value;

newNode->next = head;

head = newNode;

\*headRef = head;

}

void insertAtEnd(struct node\* head, int value){

struct node \*newNode;

newNode = malloc(sizeof(struct node));

newNode->data = value;

newNode->next = NULL;

struct node \*temp = head;

while(temp->next != NULL){

temp = temp->next;

}

temp->next = newNode;

}

void deleteFromFront(struct node\*\* headRef){

struct node\* head = \*headRef;

head = head->next;

\*headRef = head;

}

void deleteFromEnd(struct node\* head){

struct node\* temp = head;

while(temp->next->next!=NULL){

temp = temp->next;

}

temp->next = NULL;

}

void deleteFromMiddle(struct node\* head, int position){

struct node\* temp = head;

int i;

for(i=2; inext != NULL) {

temp = temp->next;

}

}

temp->next = temp->next->next;

}

int main() {

/\* Initialize nodes \*/

struct node \*head;

struct node \*one = NULL;

struct node \*two = NULL;

struct node \*three = NULL;

/\* Allocate memory \*/

one = malloc(sizeof(struct node));

two = malloc(sizeof(struct node));

three = malloc(sizeof(struct node));

/\* Assign data values \*/

one->data = 1;

two->data = 2;

three->data = 3;

/\* Connect nodes \*/

one->next = two;

two->next = three;

three->next = NULL;

/\* Save address of first node in head \*/

head = one;

display(head); // 1 --->2 --->3 --->

insertAtFront(&head, 4);

display(head); // 4 --->1 --->2 --->3 --->

deleteFromFront(&head);

display(head); // 1 --->2 --->3 --->

insertAtEnd(head, 5);

display(head); // 1 --->2 --->3 --->5 --->

deleteFromEnd(head);

display(head); // 1 --->2 --->3 --->

int position = 3;

insertAtMiddle(head, position, 10);

display(head); // 1 --->2 --->10 --->3 --->

deleteFromMiddle(head, position);

display(head); // 1 --->2 --->3 --->

}

The output of the above program is

List elements are -

1 --->2 --->3 --->

List elements are -

4 --->1 --->2 --->3 --->

List elements are -

1 --->2 --->3 --->

List elements are -

1 --->2 --->3 --->5 --->

List elements are -

1 --->2 --->3 --->

List elements are -

1 --->2 --->10 --->3 --->

List elements are -