Singly Linked List

A linked list is a linear data structure used to store elements of the same data type but not in contiguous memory locations. It is a collection of nodes where each node contains a data field and a next pointer indicating the address of the next node. So only the current node in the list knows where the next element of the list is stored. In this article, we will learn how to implement a singly linked list in C.

Implementation of Singly Linked List in C

A singly linked list is a type of linked list where only the address of the next node is stored in the current node along with the data field and the last node in the list contains NULL pointer. This makes it impossible to find the address of the particular node in the list without accessing the node previous to it. So we can only access the data sequentially in the node.

Representation of Singly Linked List in C

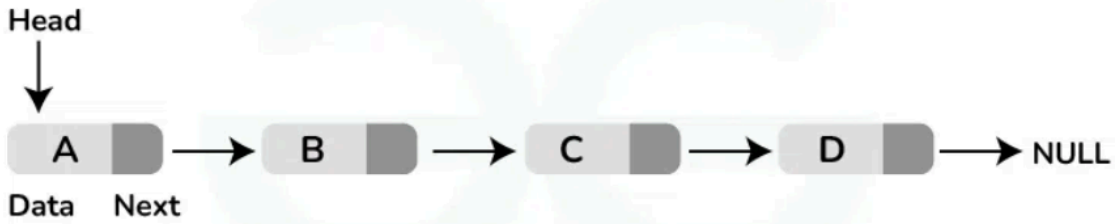
To represent a node of a singly linked list in C, we will use a structure that will have two data members **data** and **next** where:

```
struct Node {
        int data;
        Node* next;
};
```

where,

- **data:** indicates the value stored in the node.
- **next:** is a pointer that will store the address of the next node in the sequence.

The following diagram represents the structure of a singly linked list:



The pointer to the first node of the list is called head.

Insertion Operation Implementation

1. Algorithm for insertAtFirst Function

Following is the algorithm to insert a node at the start of the singly linked list:

- *Create a new node.*
- *If the linked list is empty set the new node as the **Head** and return.*
- *Connect the next pointer of this new node to the **Head** of the linked list.*
- *Update the **Head** pointer and make it points to the new node*

2. Algorithm for insertAtPosition Function

Following is the algorithm to insert a node at a specific position of the singly linked list:

- *If it is 0, call the insertAtFirst function to insert the node at the first position of the list.*

- *Check if the position is 0.*
- *Initialize a counter variable and a temporary pointer to traverse the linked list.*
- *Iterate over the linked list to find the node before the insertion point (position – 1).*
    - *If the temporary pointer becomes NULL before reaching the desired position, the position is out of range. Return.*
- *Create a new node.*
- *Point the next pointer of the new node to the node present just after the temporary pointer.*
- *Point the next pointer of the temporary node to the new node and return.*

3. Algorithm for insertAtEnd Function

Following is the algorithm to insert a node at the end of the singly linked list:

- *Create a new Node.*
- *If the list is empty, update the Head pointer to be this new node and then return.*
- *Otherwise traverse till the last node of the singly linked list.*
- *Connect next pointer of the last node to the new node.*

Deletion Operations in a Singly Liked List in C

1. Algorithm for deleteFromFirst Function

Following is the algorithm to delete the first node of the singly linked list:

1. *Ensure that the Head of the linked list is not NULL; if it is, the list is empty, so return.*
2. *Create a temporary pointer and point it to the current Head of the list.*
3. *Update the current head of the singly linked list to the next node.*
4. *Point the next pointer of the temporary node to NULL to detach it from the singly linked list.*
5. *Delete the temporary node.*

2. Algorithm for deleteFromPosition Function

Following is the algorithm to delete a node from a specific position in a linked list:

- *If it is NULL, the linked list is empty, so return.*
- *Check if the head pointer of the linked list is NULL.*
- *Check if the position is 0.*
    - *If it is 0, call the deleteFromFirst function to delete the first node.*
- *Initialize a counter variable and a temporary pointer to traverse the linked list.*

- *Iterate the linked list to find the node before the deletion point (position – 1).*
- *If the temporary pointer becomes NULL before reaching the desired position the position is out of range. Return*
- *Store the next node of the temporary pointer in a temporary pointer.*
- *Update the next pointer of the temporary pointer to the next pointer of the node to be deleted.*
- *Delete the node represented by the temporary pointer.*

3. Algorithm for deleteFromEnd Function

Following is the algorithm to delete the last node of the linked list:

- *Ensure that the Head of the linked list is not NULL; if it is, the list is empty, so return.*
- *If the singly linked list has only one node, delete the head node and point the head pointer to NULL.*
- *Traverse till the second last node of the singly linked list.*
- *Store the next node of the second last node in a temporary pointer.*
- *Connect the next pointer of the second last node to NULL.*
- *Delete the last node represented by the temporary pointer.*

Algorithm for Print Function

Following is the algorithm to iterate through the singly linked list and print the value of data present in each node:

- *Check if the HEAD of the singly linked is is NULL or not. If NULL return back.*
- *Set a temp pointer to the head of the singly linked list.*
- *While temp pointer != NULL:*
    - *Print temp->data.*
    - *Move temp to temp->next*

C Program to Implement Singly Linked List

The following program illustrates how we can implement a singly linked list in C:

```c
// // C Program for Implementation of Singly Linked List
#include <stdio.h>
#include <stdlib.h>

// Define the Node structure
struct Node {
    int data;
    struct Node* next;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new element at the beginning of the singly linked list
```

```c
void insertAtFirst(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

// Function to insert a new element at the end of the singly linked list
void insertAtEnd(struct Node** head, int data) {
    struct Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        return;
    }
    struct Node* temp = *head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
}

// Function to insert a new element at a specific position in the singly
linked list
void insertAtPosition(struct Node** head, int data, int position) {
    struct Node* newNode = createNode(data);
    if (position == 0) {
        insertAtFirst(head,data);
        return;
    }
    struct Node* temp = *head;
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL) {
        printf("Position out of range\n");
        free(newNode);
        return;
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

// Function to delete the first node of the singly linked list
void deleteFromFirst(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
```

```
    struct Node* temp = *head;
    *head = temp->next;
    free(temp);
}

// Function to delete the last node of the singly linked list
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        return;
    }
    while (temp->next->next != NULL) {
        temp = temp->next;
    }
    free(temp->next);
    temp->next = NULL;
}

// Function to delete a node at a specific position in the singly linked
list
void deleteAtPosition(struct Node** head, int position) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node* temp = *head;
    if (position == 0) {
        deleteFromFirst(head);
        return;
    }
    for (int i = 0; temp != NULL && i < position - 1; i++) {
        temp = temp->next;
    }
    if (temp == NULL || temp->next == NULL) {
        printf("Position out of range\n");
        return;
    }
    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}
```

```c
// Function to print the LinkedList
void print(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Driver Code
int main() {
    struct Node* head = NULL;

    insertAtFirst(&head, 10);
    printf("Linked list after inserting the node:10 at the beginning \n");
    print(head);

    printf("Linked list after inserting the node:20 at the end \n");
    insertAtEnd(&head, 20);
    print(head);

    printf("Linked list after inserting the node:5 at the end \n");
    insertAtEnd(&head, 5);
    print(head);

    printf("Linked list after inserting the node:30 at the end \n");
    insertAtEnd(&head, 30);
    print(head);

    printf("Linked list after inserting the node:15 at position 2 \n");
    insertAtPosition(&head, 15, 2);
    print(head);

    printf("Linked list after deleting the first node: \n");
    deleteFromFirst(&head);
    print(head);

    printf("Linked list after deleting the last node: \n");
    deleteFromEnd(&head);
    print(head);

    printf("Linked list after deleting the node at position 1: \n");
    deleteAtPosition(&head, 1);
    print(head);
```

```
    return 0;
}
```

Output

```
Linked list after inserting the node:10 at the beginning
10 -> NULL
Linked list after inserting the node:20 at the end
10 -> 20 -> NULL
Linked list after inserting the node:5 at the end
10 -> 20 -> 5 -> NULL
Linked list after inserting the node:30 at the end
10 -> 20 -> 5 -> 30 -> NULL
Linked list after inserting the node:15 at position 2
10 -> 20 -> 15 -> 5 -> 30 -> NULL
Linked list after deleting the first node:
20 -> 15 -> 5 -> 30 -> NULL
Linked list after deleting the last node:
20 -> 15 -> 5 -> NULL
Linked list after deleting the node at position 1:
20 -> 5 -> NULL
```