Show Course Contents ⊕

Overview Q&A Downloads Announcements

Delete a Node from Linked List (C Code For Deletion From Beginning, End, Specified Position & Key)

Now that we have a thorough understanding of all the cases that we will encounter when deleting an existing node from a linked list, we can code each one in C language.

Before we code, let's recall all the cases:

1. Deleting the first node -> Time complexity: O(1)
2. Deleting a node in between -> Time complexity: O(n)
3. Deleting the last node -> Time complexity: O(n)
4. Deleting the element with a given value from the linked list -> Time complexity: O(n)

Now, let's move on to the coding part. I have attached the snippet below. Refer to it while understanding the steps.

Understanding the snippet below:

1. You should have a good understanding of how to declare struct Nodes and traverse linked lists by now.
2. So, the first thing would be to create a struct *Node* and create the *linkedlistTraversal* function.
3. Do include the header file <stdlib.h>, since we'll use malloc to reserve memory locations.
4. Similar to what we did in the insertion video, create the four(choose any number) nodes. Request the memory location for each of these nodes from the heap via malloc. Link these nodes using the arrow operator.
5. And this is how we create a linked list of size 4. Let's see the cases of deletion.

Deleting the first node :

1. Create a struct Node* function *deleteFirst* which will return the pointer to the new head after deleting the current head.
2. We'll pass the current head pointer in the function.
3. Create a new struct Node* pointer *ptr*, and make it point to the current head.
4. Assign head to the next member of the list, by head = head->next, because this is going to be the new head of the linked list.
5. Free the pointer *ptr.* And return the head.

```
// Case 1: Deleting the first element from the linked list
struct Node * deleteFirst(struct Node * head){
    struct Node * ptr = head;
    head = head->next;
    free(ptr);
    return head;
}
```

Code Snippet 1: Deleting the first node

Deleting a node in between:

1. Create a struct Node* function *deleteAtIndex* which will return the pointer to the head.
2. In the function, we'll pass the current head pointer and the index where the node is to be deleted.
3. Create a new struct Node* pointer *p* pointing to *head*.

4. Create a new struct Node* pointer q pointing to *head->next*, and run a loop until this pointer reaches the index, from where we are deleting the node.
5. Assign q->next to the next member of the p structure using p-> next = q->next.
6. Free the pointer q, because it has zero connections with the list now.
7. Return head.

```
// Case 2: Deleting the element at a given index from the linked lis
struct Node * deleteAtIndex(struct Node * head, int index){
    struct Node *p = head;
    struct Node *q = head->next;
    for (int i = 0; i < index-1; i++)
    {
        p = p->next;
        q = q->next;
    }

    p->next = q->next;
    free(q);
    return head;
}
```

Code Snippet 2: Deleting a node in between

Deleting the last node :

1. Deleting the last node is quite similar to deleting from any other index. The difference holds in the limit of the while loop. Here we run a loop until the pointer reaches the end and points to NULL.
2. Assign NULL to the next member of the p structure using p-> next = NULL.
3. Break the connection between q and NULL by freeing the ptr q.
4. Return head.

```
// Case 3: Deleting the last element
struct Node * deleteAtLast(struct Node * head){
    struct Node *p = head;
    struct Node *q = head->next;
```

```
        while(q->next !=NULL)
        {
            p = p->next;
            q = q->next;
        }

        p->next = NULL;
        free(q);
        return head;
    }
```

Code Snippet 3: Deleting the last node

Deleting the element with a given value from the linked list :

1. Here, we already have a value that needs to be deleted from the list. The main thing is that we'll delete only the first occurrence.
2. Create a struct Node* function *deleteByValue* which will return the pointer to the head.
3. Pass into this function the head node, the value which needs to be deleted.
4. Create a new struct Node* pointer *p* pointing to the head.
5. Create another struct Node* pointer q pointing to the next of head.
6. Run a while loop until the pointer q encounters the given value or the list finishes.
7. If it encounters the value, delete that node by making p point the next node, skipping the node q. And free q from memory.
8. And if the list just finishes, it means there was no such value in the list. Continue without doing anything.
9. Return head.

```
    // Case 4: Deleting the element with a given value from the linked l
    struct Node * deleteByValue(struct Node * head, int value){
        struct Node *p = head;
        struct Node *q = head->next;
        while(q->data!=value && q->next!= NULL)
        {
            p = p->next;
            q = q->next;
        }

        if(q->data == value){
            p->next = q->next;
            free(q);
        }
        return head;
    }
```

Code Snippet 4: Deleting the element with a given value from the linked list

So, this was all about deletion in the linked list data structure.Here is the whole source code:

Copy

```
#include <stdio.h>
#include <stdlib.h>

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

void linkedListTraversal(struct Node *ptr)
{
    while (ptr != NULL)
    {
        printf("Element: %d\n", ptr->data);
        ptr = ptr->next;
    }
}

// Case 1: Deleting the first element from the linked list
struct Node * deleteFirst(struct Node * head){
    struct Node * ptr = head;
```

```
        head = head->next;
        free(ptr);
        return head;
    }

    // Case 2: Deleting the element at a given index from the linked list
    struct Node * deleteAtIndex(struct Node * head, int index){
        struct Node *p = head;
        struct Node *q = head->next;
        for (int i = 0; i < index-1; i++)
        {
            p = p->next;
            q = q->next;
        }

        p->next = q->next;
        free(q);
        return head;
    }

    // Case 3: Deleting the last element
    struct Node * deleteAtLast(struct Node * head){
        struct Node *p = head;
        struct Node *q = head->next;
        while(q->next !=NULL)
        {
            p = p->next;
            q = q->next;
        }

        p->next = NULL;
        free(q);
        return head;
    }
```

```
// Case 4: Deleting the element with a given value from the linked lis
struct Node * deleteAtIndex(struct Node * head, int value){
    struct Node *p = head;
    struct Node *q = head->next;
    while(q->data!=value && q->next!= NULL)
    {
        p = p->next;
        q = q->next;
    }

    if(q->data == value){
        p->next = q->next;
        free(q);
    }
    return head;
}
int main()
{
    struct Node *head;
    struct Node *second;
    struct Node *third;
    struct Node *fourth;

    // Allocate memory for nodes in the linked list in Heap
    head = (struct Node *)malloc(sizeof(struct Node));
    second = (struct Node *)malloc(sizeof(struct Node));
    third = (struct Node *)malloc(sizeof(struct Node));
    fourth = (struct Node *)malloc(sizeof(struct Node));
```

```
    // Link second and third nodes
    second->data = 3;
    second->next = third;
    // Link third and fourth nodes
    third->data = 8;
    third->next = fourth;

    // Terminate the list at the third node
    fourth->data = 1;
    fourth->next = NULL;

    printf("Linked list before deletion\n");
    linkedListTraversal(head);

    // head = deleteFirst(head); // For deleting first element of the
    // head = deleteAtIndex(head, 2);
    head = deleteAtLast(head);
    printf("Linked list after deletion\n");
    linkedListTraversal(head);

    return 0;
}
```

Previous Next

CodeWithHarry