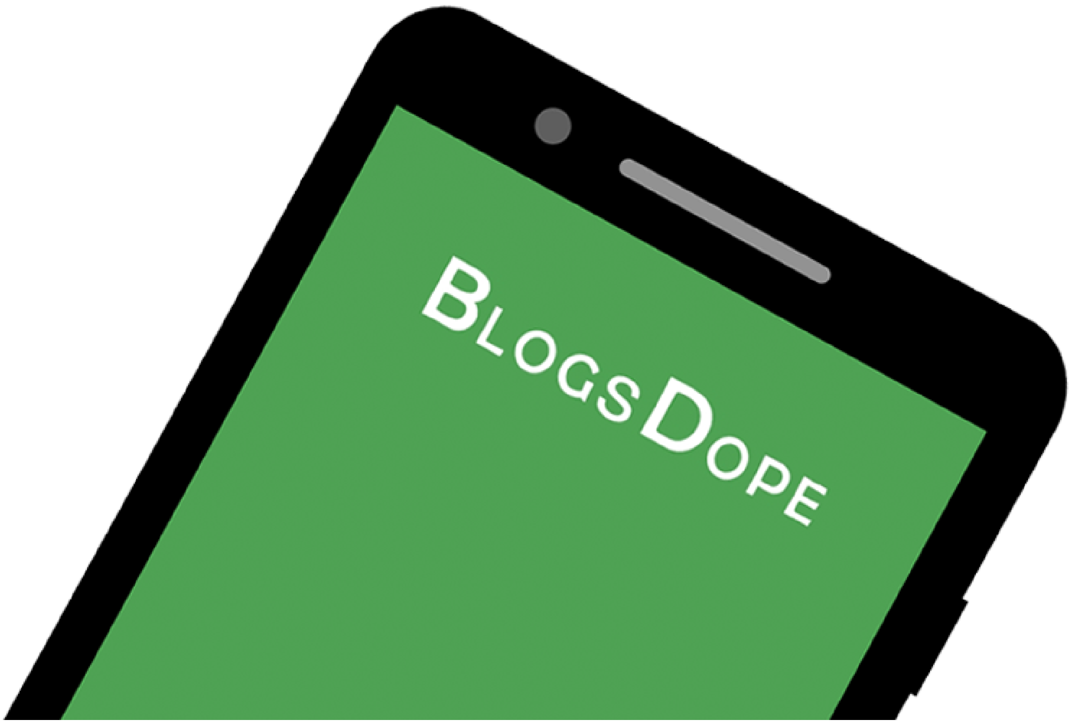
DOPE (/blog/)

Linked list traversal using loop and recursion in c++

May 30, 2017 C++ (/blog/tag/cpp/?tag=cpp) LINKED LIST (/blog/tag/linked-list/?tag=linked-list) DATA STRUCUTRE (/blog/tag/data-strucutre/?tag=data-strucutre) LOOP (/blog/tag/loop/?tag=loop) 22454



(https://play.google.com/store/apps/details?id=com.blogsdope&pcampaignid=MKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1)

Previous:

1. Linked lists in C++ (Singly linked list) (https://www.codesdope.com/blog/article/c-linked-lists-in-c-singly-linked-list/)

Make sure that you are familiar with the concepts explained in the post(s) mentioned above before proceeding further.

In the previous post, we made a linked list using the code given below.

```
#include <iostream>

using namespace std;

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

class linked_list
{
private:
    node *head,*tail;
public:
    linked_list()
    {
        head = NULL;
        tail = NULL;
    }

    void add_node(int n)
    {
        node *tmp = new node;
        tmp->data = n;
        tmp->next = NULL;

        if(head == NULL)
        {
            head = tmp;
            tail = tmp;
        }
        else
        {
            tail->next = tmp;
            tail = tail->next;
        }
    }
};

int main()
{
    linked_list a;
    a.add_node(1);
    a.add_node(2);
    return 0;
}
```

In this post, we will traverse through each node in the linked list with a loop and also with recursion.

Traversal means "visiting" or examining each node of the list. We start from the beginning and visit one node at a time until the end of the list (until the 'next' is NULL). As discussed in the previous post, we need the first element (head) to reach to any element of the list. So, we will do the traversal using the 'head' and print its element and then proceed to the next element of the list.

Thus, the steps for the traversal of the linked list are:

1. Check if the element is not NULL.
2. If it is not, then print its 'data'.
3. Change the element to the element stored in the 'next'.

And the code representing the above steps is:

```
while(p != NULL)
{
    printf("%d\n",p->data);
    p = p->next;
}
```

Here, we are first checking if the node 'p' is not NULL then we are printing the 'data' stored in it. And then changing the p to the element stored in the 'next'.

The overall code for linked traversal is:

```
#include <iostream>

using namespace std;

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

class linked_list
{
private:
    node *head,*tail;
public:
    linked_list()
    {
        head = NULL;
        tail = NULL;
    }

    void add_node(int n)
    {
        node *tmp = new node;
        tmp->data = n;
        tmp->next = NULL;

        if(head == NULL)
        {
            head = tmp;
            tail = tmp;
        }
        else
        {
            tail->next = tmp;
            tail = tail->next;
        }
    }

    void display()
    {
        node *tmp;
        tmp = head;
        while (tmp != NULL)
        {
            cout << tmp->data << endl;
            tmp = tmp->next;
        }
    }
};

int main()
{
    linked_list a;
    a.add_node(1);
    a.add_node(2);
    a.display();
    return 0;
}
```

Traversal using recursion

We can also traverse the linked list using recursion. The same logic is also used in traversing using recursion with just a difference that we will use recursion instead of the while loop. Let’s see it in action.

```
#include <iostream>

using namespace std;

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

class linked_list
{
private:
    node *head,*tail;
public:
    linked_list()
    {
        head = NULL;
        tail = NULL;
    }

    void add_node(int n)
    {
        node *tmp = new node;
        tmp->data = n;
        tmp->next = NULL;

        if(head == NULL)
        {
            head = tmp;
            tail = tmp;
        }
        else
        {
            tail->next = tmp;
            tail = tail->next;
        }
    }

    node* gethead()
    {
        return head;
    }

    void display(node *head)
    {
        if(head == NULL)
        {
            cout << "NULL" << endl;
        }
        else
        {
            cout << head->data << endl;
            display(head->next);
        }
    }
};
```

```
int main()
{
    linked_list a;
    a.add_node(1);
    a.add_node(2);
    a.display(a.gethead());
    return 0;
}
```

Here, we have declared a new function ‘gethead’ which is returning the head of the linked list and then we are passing this head to the ‘display’ function.

In this code, we have first checked if the node is NULL or not. If it is NULL, then we just printed “NULL” and when it is not then we printed the value of the ‘data’ and called the function ‘display’ again with the node next to it (‘head->next’).

Next:

1. Concatenating two linked lists in C++ (https://www.codesdope.com/blog/article/c-concatenating-two-linked-lists-in-c/)

2. Inserting a new node in a linked list in C++ (https://www.codesdope.com/blog/article/inserting-a-new-node-to-a-linked-list-in-c/)

3. Deletion of a given node from a linked list in C++ (https://www.codesdope.com/blog/article/c-deletion-of-a-given-node-from-a-linked-list-in-c/)