Show Course Contents

Overview Q&A Downloads Announcements

Linked List Data Structure: Creation and Traversal in C Language

In the last tutorial, we saw the differences between a linked list and an array. We saw the advantages and the limitations of a linked list. Today, we'll cover more on a linked lists' creation and learn how to traverse through it. If you haven't already, I recommend that you first go through the last tutorial.

I would anyway want to point some important things we learned about linked lists:

1. These are stored in non-contiguous memory locations.
2. Insertion and deletion in a linked list are very efficient in comparison to arrays.
3. An element called node holds the value as well as a pointer to the next element.

We can now move onto coding them. I've attached the snippet below for your referral. Follow them while understanding the same.

Understanding the snippet below:

1. An element in a linked list is a *struct Node.* It is made to hold integer *data* and a pointer of data type *struct Node*,* as it has to point to another *struct Node.*
2. We'll create the below illustrated linked list.

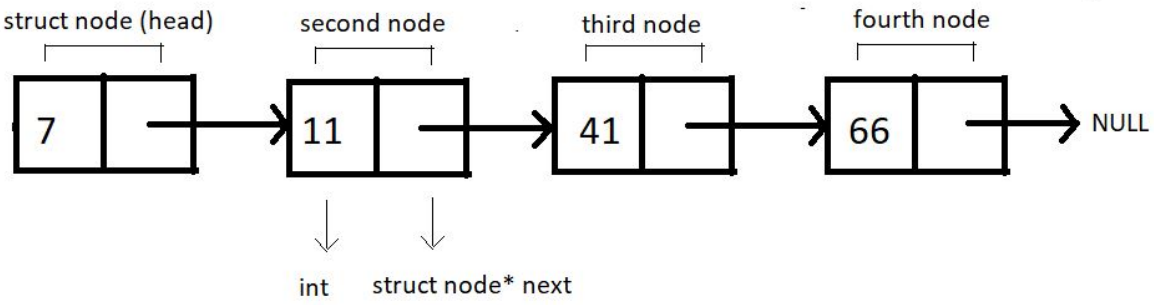

Figure 1: Illustration of the below implemented linked list.

3. We will always create individual nodes and link them to the next node via the arrow operator '⟶'.
4. First, we'll define a structure *Node* and create two of its members, an int variable *data,* to store the current node's value and a struct node* pointer variable *next.*

5. Now, we can move on to our main() and start creating these nodes. We'll name the first node, *head.* Define a pointer to head node by *struct node* head.* And similarly for the other nodes. Request the memory location for each of these nodes from heap via malloc using the below snippet.

```
head = (struct Node *)malloc(sizeof(struct Node));
```

6. Link these nodes using the arrow operator and call the traversal function.
7. Create a void function *linkedlistTraversal* and pass into it the pointer to the head node.
8. Run a while loop while the pointer doesn't point to a NULL. And keep changing the pointer *next* each time you are done printing the data of the current node.

```
#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;
    }
}

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 first and second nodes
    head->data = 7;
    head->next = second;

    // Link second and third nodes
    second->data = 11;
    second->next = third;

    // Link third and fourth nodes
    third->data = 41;
```

```
    third->next = fourth;

    // Terminate the list at the third node
    fourth->data = 66;
    fourth->next = NULL;

    linkedListTraversal(head);
    return 0;
}
```

Code Snippet 1: Creating and traversing in a linked list

Let's check if it works all fine. Refer to the output below.

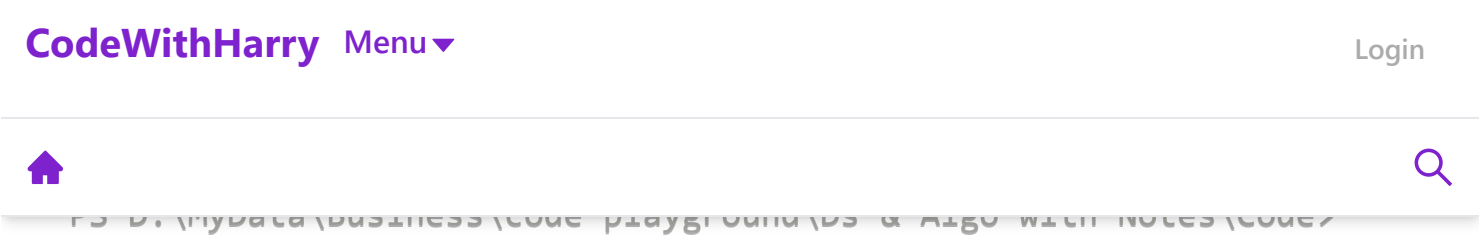

Figure 2: Output of the above program

So, this was successfully creating and traversing through the linked list. We have more things to learn ahead, but before that, make sure you feel confident about the things we have covered so far. I'll take care of the rest.
Thank you for being with me throughout. I hope you enjoyed the tutorial. If you appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial, where we'll explore more of the functionalities of linked lists. Till then, keep learning.

Previous Next

CodeWithHarry