Show Course Contents

Overview Q&A Downloads Announcements

Insertion of a Node in a Linked List Data Structure

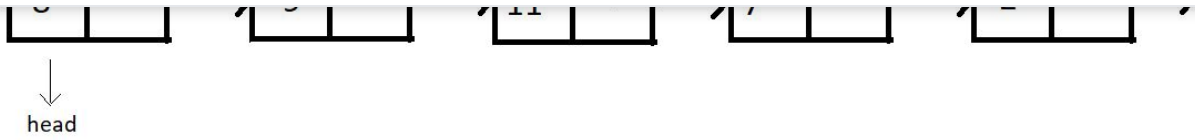
In the last tutorial, we had learned about creating a linked list using C structures and traversing through them while printing the values at each node. Today, we'll learn how to insert a node at some position and how that is more efficient than inserting an element in an array.

Inserting in an array has already been covered, and the following remarks were made:

1. A void has to be made to insert an element.
2. Creating a void causes the rest of the elements to shift to their adjacent right.
3. Time complexity: O(no. of elements shifted)

Inserting in a linked list:

Consider the following Linked List,



Insertion in this list can be divided into the following categories:

Case 1: Insert at the beginning

Case 2: Insert in between

Case 3: Insert at the end

Case 4: Insert after the node

For insertion following any of the above-mentioned cases, we would first need to create that extra node. And then, we overwrite the current connection and make new connections. And that is how we insert a new node at our desired place.

Syntax for creating a node:

```
struct Node *ptr = (struct Node*) malloc (sizeof (struct Node))
```

The above syntax will create a node, and the next thing one would need to do is set the data for this node.

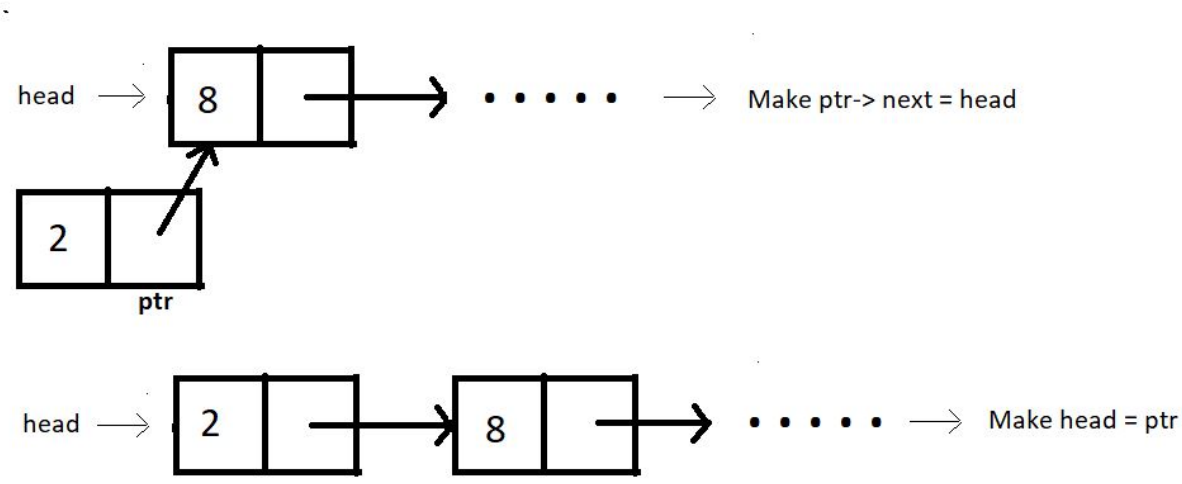
```
ptr -> data = 9
```

This will set the data.

Now, let's begin with each of these cases of insertion.

Case 1: Insert at the beginning

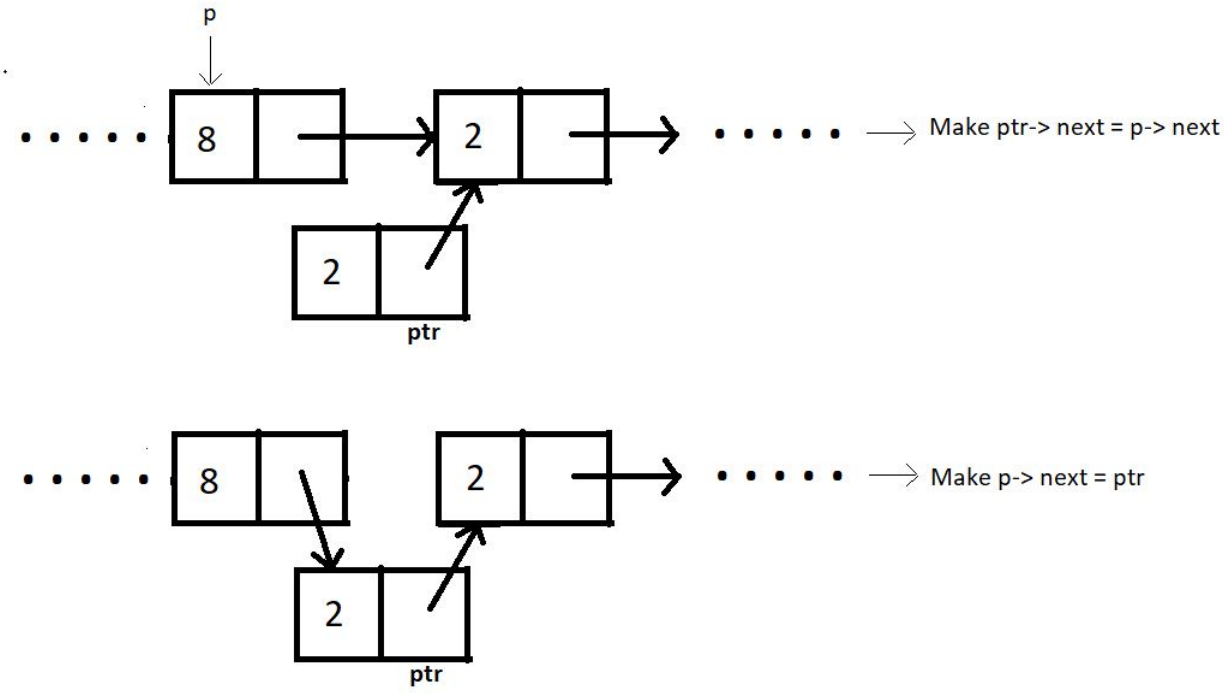
In order to insert the new node at the beginning, we would need to have the head pointer pointing to this new node and the new node's pointer to the current head.



Case 2: Insert in between:

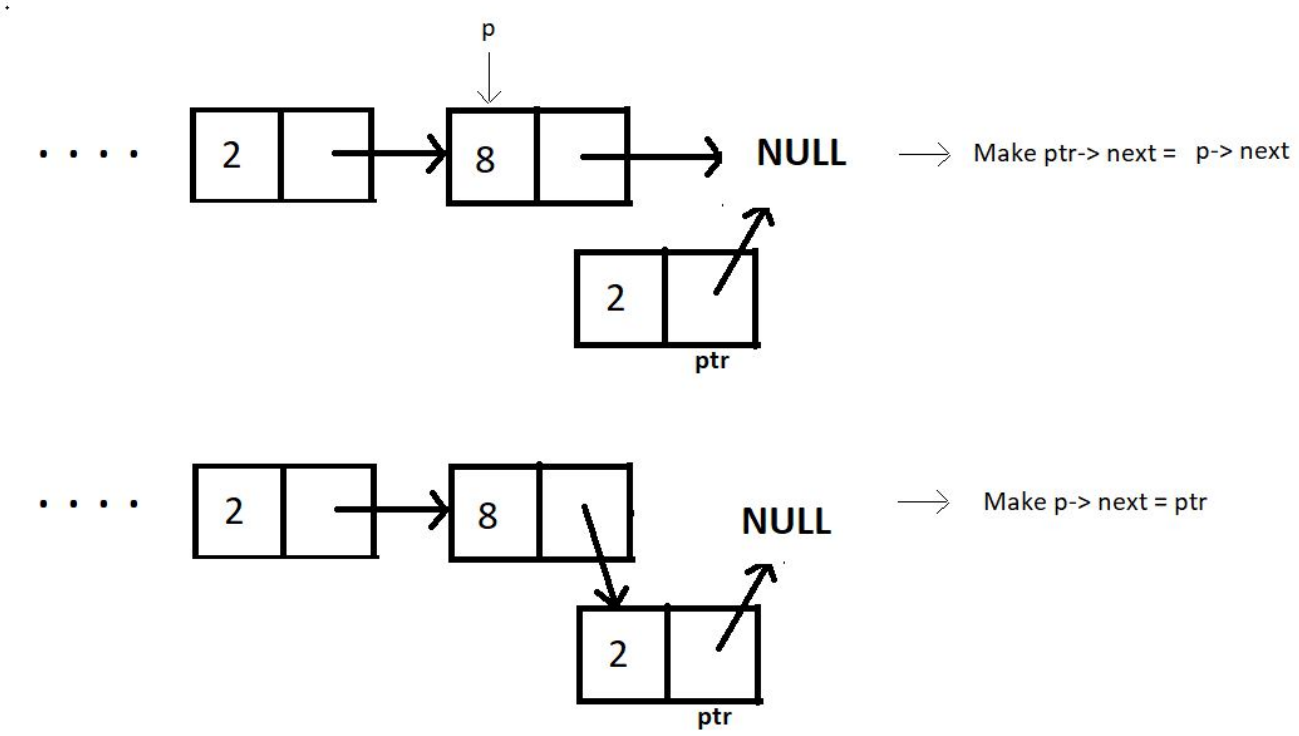
Assuming index starts from 0, we can insert an element at index i>0 as follows:

1. Bring a temporary pointer p pointing to the node before the element you want to insert in the linked list.
2. Since we want to insert between 8 and 2, we bring pointer p to 8.



Case 3: Insert at the end:

In order to insert an element at the end of the linked list, we bring a temporary pointer to the last element.

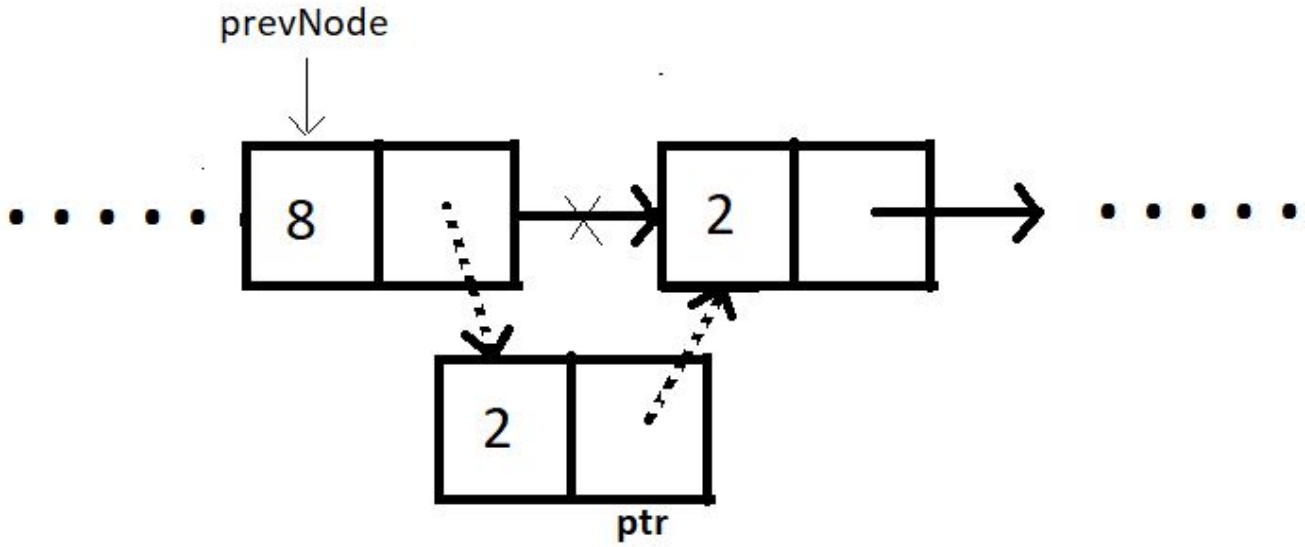


Case 4: Insert after a node:

Similar to the other cases, ptr can be inserted after a node as follows:

ptr->next = prevNode-> next;

prevNode-> next = ptr;



Summarizing, inserting at the beginning has the time complexity O(1), and inserting at some node in between puts the time complexity O(n) since we have to go through the list to reach that particular node. Inserting at the end has the same time complexity O(n) as that of inserting in between. But if we are given the pointer to the previous node where we want to insert the new node, it would just take a constant time O(1).

You must have had no problem understanding all these manipulations if you have followed me from the beginning. It must have been a cakewalk for you. Try to implement them on your own IDEs, and continue going through this until you finish with the concepts. We are anyway going to learn their codes in the next tutorial.

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 try to code these inserting methods. Till then, keep learning.

Previous Next

CodeWithHarry