# Linked-List using C Part 2

## Learning step 3 : Prepend element

Prepending is just figuring out the head and next pointers to switch between current head node and new node.

```c
/*Prepend needs to take list and val as parameters
 * create a node and setnew node's next to list's head which is node 1
 * set the head node to point to nnode
*/
void prepend(linkedlist* list, int val){
        Node* nnode = createnode(val);
        nnode->next = list->head;
        list->head = nnode;
}
```

```c
int main(){
        linkedlist* list = createlist();
        append(list, 10);
        append(list, 20);
        append(list, 30);
        prepend(list, 0);
        traverse(list);
        return 0;
}
```

0 is appended in output as the head

```
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ gcc -g linked_list.c -o linkedl
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ ./linkedl
0 -> 10 -> 20 -> 30 -> NULL
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ vim linked_list.c
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$
```

## Learning step 4 : insert at index function

Iterate till just before the exact target and then just set the newly created niode's next pointer to tnode's next which was the last node previous to target so technically we just insert 1 before it and then tnode->next would be switched to nnode(inserted).

```c
/* count i for 1 index before the target
 * iterate till the -1 position from target
 * just switch the pointers next to newly created node
*/
void insertAt(linkedlist* list, int index, int val){
        int i = 0;
        Node* tnode = list->head;
        while(i<index-1){
        tnode = tnode->next;
        i++;
        }
        Node* nnode = createnode(val);
        nnode->next = tnode->next;
        tnode->next = nnode;


}
```

```c
int main(){
        linkedlist* list = createlist();
        append(list, 10);
        append(list, 20);
        append(list, 30);
        prepend(list, 0);
        insertAt(list, 2, 15);
        traverse(list);
        return 0;
}
```

```
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ ./linkedl
0 -> 10 -> 15 -> 20 -> 30 -> NULL
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ vim linked_list.c
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ vim linked_list.c
```

## Learning step 5 : Delete at index function

This took me long but its just storing target to a temporrary pointer and then replacing target's previous node's next to target's next

```c
/*
 *iterate with tnode and stop just before the target
 *create a new temp pointer variable that points to target's next node
 * Lastly update current tnode;s next with cnodes's next which is target's next
 *Delete the cnode and free memory
 */


void deleteAt(Linkedlist* list, int index){
int i = 0;
Node* tnode = list->head;
while(i<index-1)
{
        tnode = tnode->next;
        i++;
}

    Node* cnode = tnode->next;
    tnode->next = cnode->next;
    free(cnode);
}
```

Call the index you want to delete. Below we have successfully deleted 20

```
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ gcc -g linked_list.c -o linked
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$ ./linkedl
0 -> 10 -> 15 -> 30 -> NULL
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/Day2$
```