Doubly Linked-List using C Part 2

Learning step 3 : Insert At index

For this we have quite a few cases to consider like if listis empty or if position is 0 and list is non empty. In both cases we add it to list basically prepending and setting it to new head.
Lastly we iterate till index and insert nnode in position by moving pointers across.

```c
/*Insert at needs list, value and index as parameter
 *First it checks if list is empty or if inserting in first position, then we set nnode -> next to head,
 *if head was not empty we just add node in front
 * Check if tail is null too and make it point to new node
 *in other case just use a tnode to iterate through list till index position
 * If we reached tail we will append to it
 *Otherwise we will set the next and previous of current tnode to nnode and then we set the neighbor pointers to point at nnode */

void insertAt(dlist *list, int index, int val){
        Node *nnode == createnode(val);

        if (list->head == NULL || index == 0){
                nnode->next = list->head;
                if (list->head != NULL)
                        list->head->prev = nnode;
                list->head = nnode;
                if (list->tail == NULL)
                        list->tail = nnode;
                return;
        }
        Node *tnode = list->head;
        int i =0;
        while(tnode != NULL && i<index-1){

                tnode = tnode->next;
                i++;
        }
        if (tnode == list->tail){
                tnode->next = nnode;
                nnode->prev = tnode;
                list->tail = nnode;
        }else
        {
                nnode->next = tnode->next;
                nnode->prev = tnode;
                tnode->next->prev = nnode;
                tnode->next = nnode;
        }


}
```

```c
int main(){
        dlist *list = createlist();
        append(list, 10);
        append(list, 20);
        append(list, 30);
        insertAt(list, 1, 15);
        traverse(list);
        return 0;
}
```

```
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/DoublyLinkedList$ ./insert
10<->15<->20<->30<->NULL
```

Learning step 4: Delete At

For delet at index first check if list->head is null and return, then we check if the iteration till index landed tnode to head or tail
So first we check if tnode's prev is not null, then we unlink tnode and make the link skip over to tnode's next, else we just set the head to tnode's next
Then we check if if position landed on tail, if no just update previous node and skip tnode else update tail to ne tnode's prev

```c
/*We check if list is empty and then return
 *Iterate through till index
 *if tnode is null return
 * if prev node was not null just unlink tnode else it is head and make it point forward
 *if tnode-> next is not null, we unlink the prev node of tnode's next  to tnode's previous
 *else if it is tail, we can set it to tnode's prev
 *free moemory for tnode once it is unlinked
  */
void deleteAt(dlist *list, int index){
        if (list->head == NULL){
        return;
        }
        Node *tnode = list->head;
        int i = 0;
        while(tnode != NULL && i<index){
                tnode = tnode->next;
                i++;
        }
        if (tnode == NULL) return;

        if (tnode->prev != NULL)
                tnode->prev->next = tnode->next;
        else
                list->head = tnode->next;
        if (tnode->next != NULL)
                tnode->next->prev = tnode->prev;
        else
                list->tail = tnode->prev;
        free(tnode);


}
```

```c
int main(){
        dlist *list = createlist();
        append(list, 10);
        append(list, 20);
        append(list, 30);
        insertAt(list, 1, 15);
        deleteAt(list, 1);
        traverse(list);
        return 0;
}
```

```
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/DoublyLinkedList$ ./delete
10<->20<->30<->NULL
```