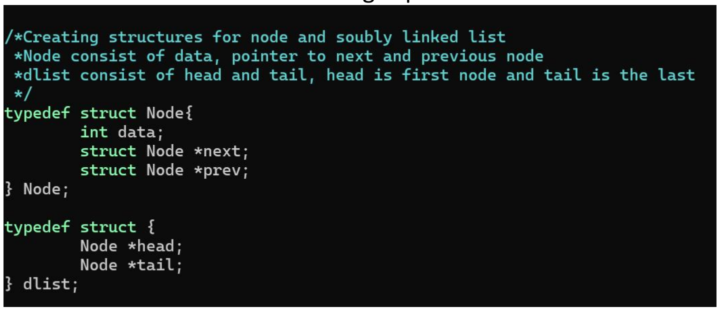
Doubly Linked-List using C Part 1

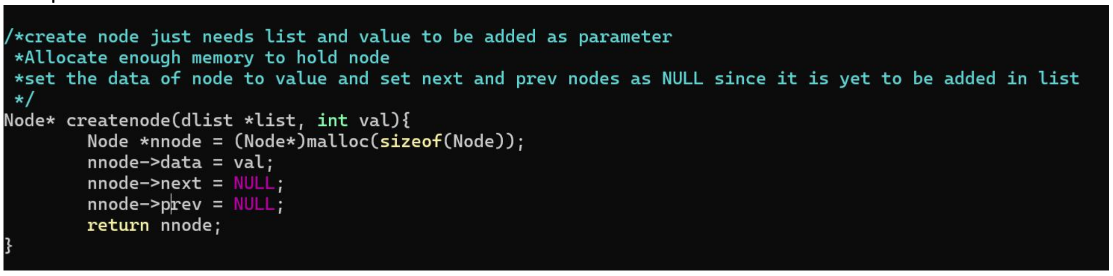
Learning step 1 : Create Nodes, doubly linked list

Create structure to allocate enough space to list and node later



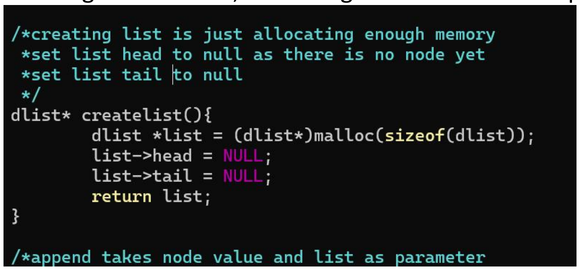
```
/*Creating structures for node and soubly linked list
 *Node consist of data, pointer to next and previous node
 *dlist consist of head and tail, head is first node and tail is the last
 */
typedef struct Node{
        int data;
        struct Node *next;
        struct Node *prev;
} Node;

typedef struct {
        Node *head;
        Node *tail;
} dlist;
```

Creating Node is same as linked-list earlier just small change which is 2 pointers to keep track of next and previous nodes.



```
/*create node just needs list and value to be added as parameter
 *Allocate enough memory to hold node
 *set the data of node to value and set next and prev nodes as NULL since it is yet to be added in list
 */
Node* createnode(dlist *list, int val){
        Node *nnode = (Node*)malloc(sizeof(Node));
        nnode->data = val;
        nnode->next = NULL;
        nnode->prev = NULL;
        return nnode;
}
```

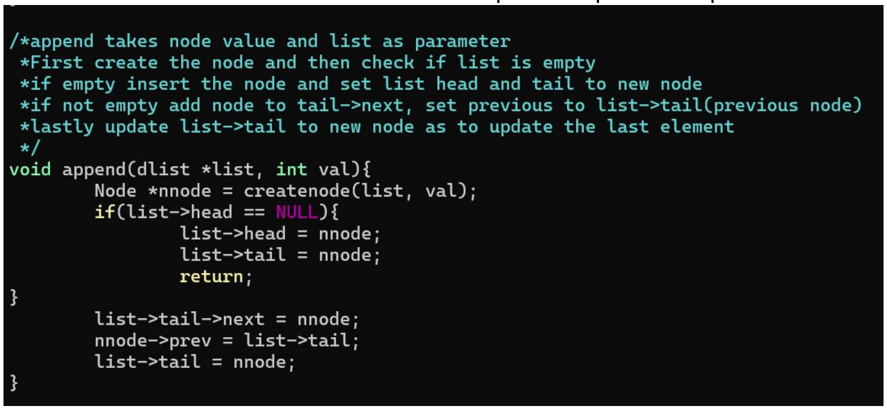
Creating list is similar, the change is to use head tail pointers to keep track of first and last element.



```
/*creating list is just allocating enough memory
 *set list head to null as there is no node yet
 *set list tail to null
 */
dlist* createlist(){
        dlist *list = (dlist*)malloc(sizeof(dlist));
        list->head = NULL;
        list->tail = NULL;
        return list;
}

/*append takes node value and list as parameter
```

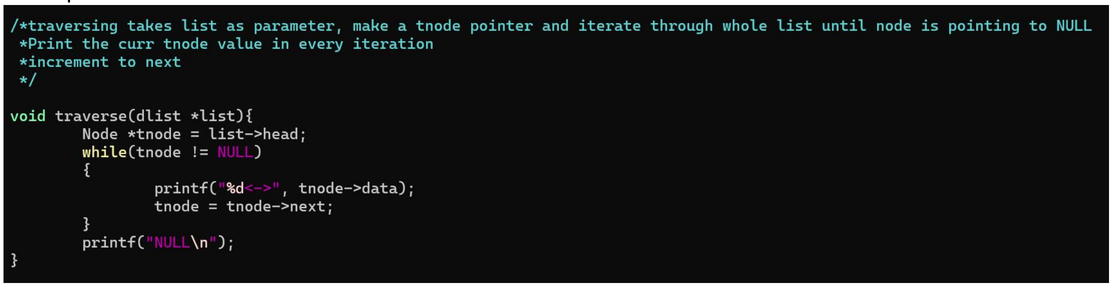
Learning step 2 : Append the List, traverse and print

Append checks for list head, if empty set the new node as head and tail. If not add the new node to tail and set the new node as tail and set the previous pointer to previous node.



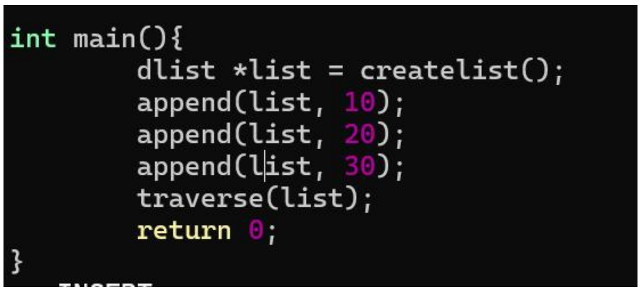
```
/*append takes node value and list as parameter
 *First create the node and then check if list is empty
 *if empty insert the node and set list head and tail to new node
 *if not empty add node to tail->next, set previous to list->tail(previous node)
 *lastly update list->tail to new node as to update the last element
 */
void append(dlist *list, int val){
        Node *nnode = createnode(list, val);
        if(list->head == NULL){
                list->head = nnode;
                list->tail = nnode;
                return;
}
        list->tail->next = nnode;
        nnode->prev = list->tail;
        list->tail = nnode;
}
```

Printing is similar just keep iterating through the list node by node by incrementing next pointer and use a pointer variable for that.

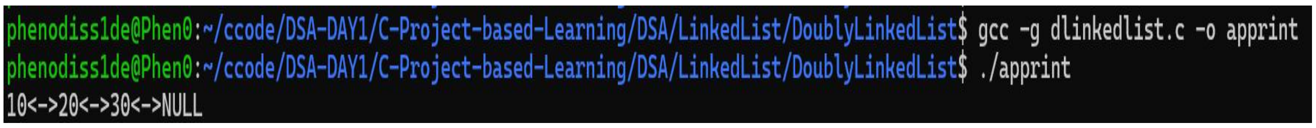


```c
/*traversing takes list as parameter, make a tnode pointer and iterate through whole list until node is pointing to NULL
 *Print the curr tnode value in every iteration
 *increment to next
 */

void traverse(dlist *list){
        Node *tnode = list->head;
        while(tnode != NULL)
        {
                printf("%d<->", tnode->data);
                tnode = tnode->next;
        }
        printf("NULL\n");
}
```

Now make the doubly linked list



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



```
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/DoublyLinkedList$ gcc -g dlinkedlist.c -o apprint
phenodiss1de@Phen0:~/ccode/DSA-DAY1/C-Project-based-Learning/DSA/LinkedList/DoublyLinkedList$ ./apprint
10<->20<->30<->NULL
```