Implementation of singly linked list

Algorithm

Insertion at index

```
1. start
2. accept a node , newNode and pos
3. if start = NULL , then set start = newNode
4. else , do
    1. set cur = 0
    2. set i = start
    3. repeat while cur < pos - 1 and i != NULL
        1. i = i->next
    4. if i->next = NULL, then set i->next = newNode
    5. else , do
        1. newNode->next = i->next
        2. i->next = newNode
    6. if pos = 0, then set start = newNode
5. stop
```

Deletion at index

```
1. start
2. accept index
3. if start = NULL, stop
4. set cur = 0 , i = start , temp = NULL
5. repeat while cur < pos - 1 and i != NULL
    1. i = i->next
6. if pos != 0 , do
    1. temp = i->next
    2. i->next = (i->next)->next
7. else ,
    1. temp = i
    2. start = i->next
8. stop
```

Searching in list

```
1. start
2. accept number n to be searched for in the list
3. set loc = 0
4. if start = NULL, stop
5. set i = start
6. repeat while i != NULL
    1. if n = i->data, do
        1. print found at loc
        2. stop
7. print not found
8. stop
```

Source Code

```c
#include<stdio.h>
#include<stdbool.h>
#include<stdlib.h>

struct NODE
{
    int num;
    struct NODE * next;
}*start=NULL;

typedef struct NODE NODE;
int count=0;

NODE* create_node(int num){
    NODE *newNode = calloc(1,sizeof(NODE));
    newNode->num = num;
    newNode->next = NULL;
    count++;
    return newNode;
}
/* inserts a node exactly at index , i.e , insertion after index-1
if the index is greater than the number of nodes in the list , the new node is
↪  appeneded to the list
prepend to list : insert at index 0
append to list : inset at index (count+1) or INFINITY
    */
void insert_at(int num,int pos)
{
    NODE *newNode = create_node(num);

    if(start==NULL){
        start=newNode;
        return;
    }
    int cur=0;
    NODE *i;
    for(i=start;cur<pos-1 && i->next ;i=i->next,cur++)
        if(!(i->next)){
            i->next=newNode;
            return;
        }
    newNode->next=i->next;
    i->next=newNode;

    if(pos==0)
        start=newNode;
}
// deletes the node at index
bool delete_at(int pos)
{
    if(!start)
        return false;
    int cur=0;
    NODE *i=start,*temp=NULL;
    for(;cur<pos-1 &&i;i=i->next,cur++);
    if(pos){
        temp=i->next;
        i->next=(i->next)->next;
    }
    else{
        temp=i;
        start=i->next;
    }

    free(temp);
    count--;
    return true;
}

// to display node
void display()
{
    for(NODE *i=start;i;i=i->next)
        printf(" %d", i->num);
}

// searching
int searching(int search)
{
    int loc=0;
    if(!start)
        return -1;
    for(NODE *i=start;i!=NULL;i=i->next,loc++)
        if(i->num==search)
            return loc;
    return -1;
}

// To free list
void freelist()
{
    NODE *temp=NULL;
    for(NODE*i=start;i!=NULL;i=i->next){
        free(temp);
        temp=i;
    }
    free(temp);
}

int main()
{
    start = NULL;
    int choice;
    int num,x;

    printf("1. Append\n2. Prepend\n3. Insert at index\n4. Delete from index \n5.
↪   Search in list \n6. Display list \n0. Exit");
    do
    {
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:{
                        printf("\nEnter number: ");
                        scanf("%d", &num);
                        insert_at(num,count+1);
                        break;
                   }
            case 2:{
                        printf("\nEnter number: ");
                        scanf("%d", &num);
                        insert_at(num,0);
                        break;
                   }
            case 3:{
                        int pos;
                        printf("\nEnter number: ");
                        scanf("%d", &num);
                        printf("Enter position: ");
                        scanf("%d", &pos);
                        insert_at(num,pos);
                        break;
                   }
            case 4:{
                        int pos;
                        printf("\nEnter position: ");
                        scanf("%d", &pos);
                        if(!delete_at(pos))
                            printf("\nList Empty!");
                        break;
                   }
            case 5:{
                        printf("\n Enter element to search : ");
                        scanf("%d",&x);
                        searching(x);
                        break;
                   }
            case 6:{
                        display();
                        break;
                   }
        }
    }while(choice!=0);
    freelist();
    return 0;
}
```