LAB 3

LINKED LIST IMPLEMENTATION IN C

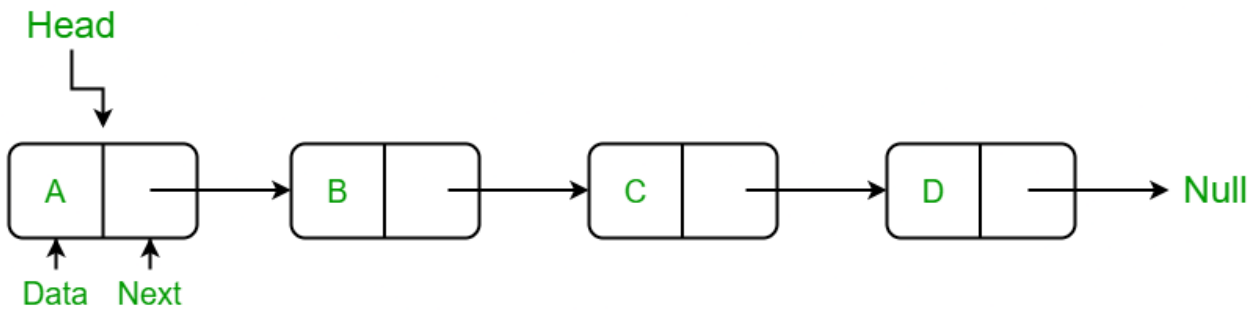
Linked lists are fundamental data structures in computer programming. They offer dynamic memory allocation and flexibility, making them valuable for various applications. In this report, we have implemented a linked list in the C programming language.

The implementation covers key aspects such as node structure, insertion and deletion algorithms, and functions for searching and displaying elements.

Key Definitions:

1. **Node:**
    - A node is the basic unit of a linked list.
    - It contains two fields: data (information to be stored) and a reference (pointer) to the next node in the sequence.
2. **Linked List:**
    - A linear data structure where elements (nodes) are connected by pointers.
    - Unlike arrays, linked lists do not have a fixed size and can dynamically grow or shrink.
3. **Head/Start:**
    - The first node in a linked list.
    - Serves as the starting point for traversing the list.
4. **Pointer:**
    - A variable that stores the memory address of another variable.
    - In linked lists, pointers are used to connect nodes and navigate through the list.
5. **Insertion:**
    - The process of adding a new node to the linked list.
    - Can occur at the beginning (prepend), end (append), or in some nth position.
6. **Deletion:**
    - Removing a node from the linked list.
    - Involves adjusting pointers of neighboring nodes to maintain the list's integrity.
7. **Traversal:**
    - Moving through the linked list to access or manipulate nodes.
    - Typically done using a loop that follows node pointers from the head to the end.
8. **Search:**
    - Locating a specific node in the linked list based on its data or position.
    - Involves traversing the list until the target node is found.
9. **Dynamic Memory Allocation:**
    - Allocating memory for nodes during runtime using functions like `malloc()` in C.
    - Ensures flexibility in memory usage for variable-sized data structures.

10. **Advantages and Disadvantages:**
    - Linked lists allow dynamic size, efficient insertions, and deletions.
    - However, they may have higher space overhead due to pointers and slower random access compared to arrays.
11. **Applications:**
    - Linked lists are used in dynamic memory allocation, implementation of stacks and queues, and certain algorithms like graph traversals.



Algorithms

1. **Node Structure:**

   Defines a structure for a linked list node containing an integer (`info`) and a pointer to the next node (`next`).

```
struct Node {
      int info;
      struct Node* next;
};
typedef struct Node node;
```

2. **Function - First Insert:**

   Inserts a node at the beginning of the linked list.

- Allocate memory for a new node.
- Set the new node's next pointer to the current start.
- Update the start pointer to the new node.

```
void f_insertnode(node* i, int d);
```

3. **Function - Last Insert:**

   Inserts a node at the end of the linked list.

- Allocate memory for a new node.
- Traverse the list to find the last node.

- Set the last node's next pointer to the new node.

```
void l_insertnode(node* i, int d);
```

4. **Function - Nth Insert:**

   Inserts a node at the nth position in the linked list.

- Allocate memory for a new node.
- Traverse the list to find the (n-1)th node.
- Set the new node's next pointer to the (n-1)th node's next, and update the (n-1)th node's next to the new node.

```
void n_insertnode(node* i, int n, int d);
```

5. **Function - First Delete:**

   Deletes the first node in the linked list.

- If the list is not empty, update the start pointer to the second node.

```
void f_deletenode();
```

6. **Function - Last Delete:**

   Deletes the last node in the linked list.

- If the list is not empty, traverse to the second last node and free the last node.

```
void l_deletenode(node* i);
```

7. **Function - Nth Delete:**

   Deletes the node at the nth position in the linked list.

- Traverse to the (n-1)th node, update its next pointer to skip the nth node, and free the nth node.

```
void n_deletenode(int n, node* i);
```

8. **Function - Search Node:**

   Searches for and displays the data at the nth position in the linked list.

- Traverse to the nth node and display its data.

```
void searchnode(int n, node* i);
```

9. **Function – Display:**

- Displays all elements in the linked list.
- Traverse the list and display each node's data.

```
void display();
```

DISCUSSION

Though this lab we learned the implementation of linked list in c programming language. Although we had studied the algorithm already, actually implementing it with the proper syntax and error conditions proved to be bit of a challenge. The traverse algorithm was took a bit of carefulness in loop design. After few tries and carefullness, the program worked sucessfully. The lab was completed as a sucessful learning experience.

CONCLUSION

We hereby conclude the lab report on linked list implementation in c programming language.

CODE:

```
//linked list
#include<stdio.h>
#include<stdlib.h>

//create structure for node
struct Node{
      int info;
      struct Node* next;
};

typedef struct Node node;

//point start node to NULL
node *start=NULL;


//inserts node at beginning
void f_insertnode(node* i, int d){
      i=(node*)malloc(sizeof(node));
            i->next=start;
            i->info=d;
            start=i;
}
//insert node at last
void l_insertnode(node* i, int d){
      i=(node*)malloc(sizeof(node));
      node* temp=(node*)malloc(sizeof(node));
      i->next=NULL;
      i->info=d;
      temp=start;
      if (start==NULL){
            start=i;
      }

      while(temp->next!=NULL){
            temp=temp->next;
      }
      temp->next=i;
```

```
}
void n_insertnode(node* i, int n, int d){
      i=(node*)malloc(sizeof(node));
      i->info=d;
      int c=1;
      if(n==1){
            i->next=start;
            start=i;
      }
      else{
      node* temp=(node*)malloc(sizeof(node));
            temp=start;
            while(c!=(n-1)){
                  temp=temp->next;
                  c++;
            }
            i->next=temp->next;
            temp->next=i;
      }

      }
//delete node
void f_deletenode(){
      node* temp=(node*)malloc(sizeof(node));
      if(start==NULL){
            printf("list empty!");
      }
      else{temp=start;
            start=temp->next;}
}
void l_deletenode(node* i){
      node* temp=(node*)malloc(sizeof(node));
      temp=start;

      if(start==NULL){
            printf("No data to delete!");
      }
      else if(temp->next==NULL){
            start=NULL;
            free(temp);
            printf("item deleted!");

      }
      else{
            while((temp->next)->next!=NULL){
                  temp=temp->next;
            }
            free(temp->next);
            temp->next=NULL;
```

```
            printf("item deleted!");
      }
}
void n_deletenode(int n, node* i){
      int c=1;
      node* temp=(node*)malloc(sizeof(node));
      temp=start;
      if(n==1){
            f_deletenode();
            printf("item deleted!");
      }

      else{
            while(c!=(n-1)){
                  temp=temp->next;
                  c++;
            }
            i->next=(temp->next)->next;
            printf("item deleted!");
      }
}

//search for data
void searchnode(int n, node* i){
      node* temp=(node*)malloc(sizeof(node));
      int c=0;

            if(start==NULL){
                  printf("list EMPTY!");
            }
            else if(n==1){
                  temp=start;
                  printf("%d", temp->info);
            }
      else{
            temp=start;
            c++;
            while(c!=(n-1)){
                  if(temp->next==NULL){
                        printf("reached the end of list!");
                        exit(0);
                  }
                  temp=temp->next;
                  c++;
            }
            printf("%d",(temp->next)->info);
      }
}
```

```
void display(){
      node* temp;
      temp=start;
      if(start==NULL){
            printf("EMPTY LIST!\n");
      }
            else{while(temp->next!=NULL){
                        printf("%d ", temp->info);
                        temp=temp->next;
                  }
                        printf("%d ", temp->info);
                  }
            }



int main(){
      int data,n;
      int c;
      node* item;
      printf("WELCOME TO THE LINKED LIST IMPLEMENTATION IN C\n");
      printf("1.add item at first  2.add item at last 3.add item at nth position
4.delete first item  5.delete last item  6.delete item at nth position  7.display
item  8.search item: ");
      scanf("%d", &c);
      do{
            switch(c){
            case 1:
                  printf("Add data to add at the beginning of list: ");
                  scanf("%d",&data);
                  f_insertnode(item, data);
                  printf("Data added!\n");
                  break;
            case 2:
                  printf("Add data to add at the end of list: ");
                  scanf("%d", &data);
                  l_insertnode(item, data);
                  printf("Data added!\n");
                  break;
            case 3:
                  printf("Type the position and the data to add: ");
                  scanf("%d %d",&n, &data);
                  n_insertnode(item, n, data);
                  printf("data added!\n");

            break;
            case 4:
                  f_deletenode();
```

```
                printf("node deleted!\n");
                break;
            case 5:
                l_deletenode(item);
                break;
            case 6:
                printf("type the position of node to delete: ");
                scanf("%d",&n);
                n_deletenode(n, item);
                break;
            case 7:
                display();
                break;
            case 8:
                printf("type the position of node to display: ");
                scanf("%d", &n);
                searchnode(n,item);
                break;
            default:
                exit(0);
                break;
            }
            printf("1.add item at first   2.add item at last 3.add item at nth
position 4.delete first item   5.delete last item   6.delete item at nth position
7.display item   8.search item: ");
            scanf("%d", &c);
      }while(c!=0);
      return 0;
}
```

output

/* NOTE: the '>>' symbol is not present in the actual output. It is added for better readibility*/

```
>>WELCOME TO THE LINKED LIST IMPLEMENTATION IN C
1.add item at first   2.add item at last 3.add item at nth position
4.delete first item   5.delete last item   6.delete item at nth
position   7.display item   8.search item:
1
Add data to add at the beginning of list: 11
Data added!
```

```
>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item:
2
Add data to add at the end of list: 22
Data added!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item:
3
Type the position and the data to add: 2 33
data added!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item:
1
Add data to add at the beginning of list: 44
Data added!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
2
Add data to add at the end of list: 55
Data added!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
3
Type the position and the data to add: 4 66
data added!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
7
44 11 33 66 22 55
```

```
>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
4
node deleted!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
5
item deleted!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
7
11 33 66 22

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
6
type the position of node to delete: 3
item deleted!

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
7
11 33 22

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item
8
type the position of node to display: 2
33

>>1.add item at first  2.add item at last 3.add item at nth position
4.delete  first  item   5.delete  last  item   6.delete  item  at  nth
position  7.display item  8.search item ^C
```