DSA LAB PROGRAMS - Linked Lists:

By Sindhu Sankati AP19110010477 CSE-G

PROGRAM 41

Title: Intializing and displaying a node of five elements

Objective:

Write a C program to create a single linked list with 5 nodes. (5 integers are taken from user input) and display the linked-list elements.

Explanation:

Pseudo Code:

>Create node function

```
/*Assume that it is a function where data is passed*/
1. SET data
2. newnode = NEW node //allocates memory to node
3. node->data = data
4. node->next = NULL
5. RETURN node
6. END
```

>Creating linked-list of 5 nodes

```
1. SET ptr=head=NULL
2. FOR i from 0 to 5
   /*for creating n nodes: i from 0 to n*/
   {
3.    READ data
4.    IF head == NULL: head = ptr = CREATENODE(data)
5.    ELSE
      {
6.        ptr->next = CREATENODE(data)
7.        ptr = ptr->next
      }
   }
8. RETURN head
9. END
```

>Displaying linked list

```
   /*head is a instance of the actual head here*/
1. WHILE head != NULL
2. {
3.    PRINT head->data
4.    head = head->next
5. }
6. END
```

Code:

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

//declaration if a node
typedef struct Node{
    int data;
    struct Node *next;

}node;

//function which creates nodes
node* createNode(int data){
    node *n = ((node*)malloc(sizeof(node)));
    n->data = data;
    n->next = NULL;
    return n;
}

//function which creates a list of 5 nodes
node* createList(){

    int n=5,data;
    node  *p, *head = NULL;

    //runs loop 5 times
    while(n--){
        printf("Enter a number\n");
        scanf("%d",&data);
        if(head == NULL){
            //intializing newnode as head
            head = createNode(data);
            p = head;
        }
        else{
        p->next = createNode(data);
```

```
            p = p->next;
            }
        }
        //return the list of 5nodes
        return head;

    }

    //displays elemets in linked list till null
    void display(node *head){
        while(head!=NULL){
            printf("%d->",head->data);
            head = head->next;
        }
        printf("NULL\n");
    }

    //main
    int main()
    {
        //driver code
        node *head=createList();
        display(head);
        return 0;
    }
```

Output:

```
b in ~/DESKTOP/Sindhu/Dsa-Lab/linked-lists
$ ./a.out
Enter a number
23
Enter a number
34
Enter a number
12
Enter a number
67
Enter a number
43
23->34->12->67->43->NULL
```

Conclusion:

We get the expectd outputs. The time complexity for this code is constant. But if insteead of 5 nodes we are to create n nodes then the time complexity for both inputting and displaying would be O(n).

PROGRAM 42

Title: Searching a element in linked list

Objective:

Write a C program to search an element in a singly-linked list.

Explanation:

When given with an integer we we search for the element in the linked list and print the position (index+1) of the element. If not found should print -1 Here we will use linear search because list might not be sorted.

Pseudo Code:

>Search

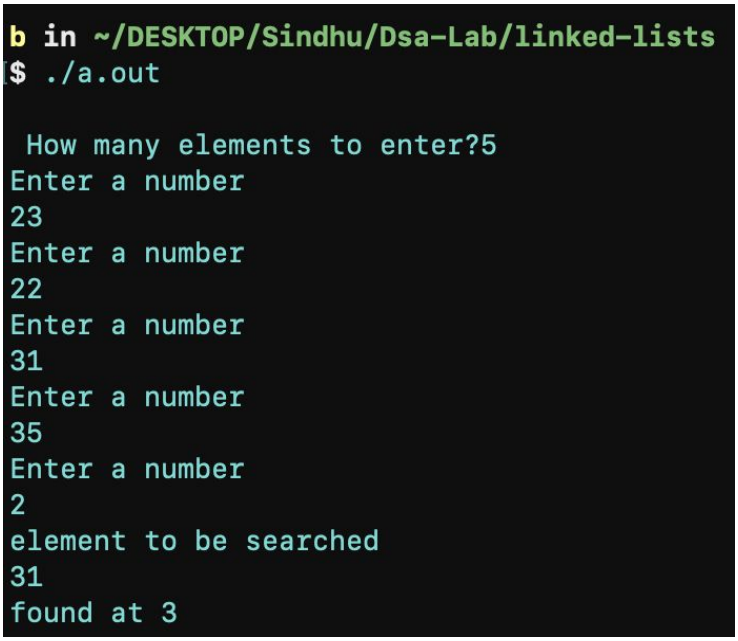
```
/*Search is the element to be searched*/

1. SET pos = 1
2. WHILE head != NULL
   {
3.     IF head->data == search   //if found
4.          RETURN pos
5.     head = head->next
6.     pos++
   }
7. RETURN -1 //element not in list
8. END
```

Code:

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

//declaration if a node
typedef struct Node{
    int data;
    struct Node *next;

}node;
//function which creates nodes
node* createNode(int data){
    node *n = ((node*)malloc(sizeof(node)));
    n->data = data;
    n->next = NULL;
    return n;
}
//function which creates a list of n nodes
node* createList(){
```

```
    int n,data;
    node  *p, *head = NULL;
    printf("\n How many elements to enter?");
    scanf("%d", &n);
    //runs loop 5 times
    while(n--){
        printf("Enter a number\n");
        scanf("%d",&data);
        if(head == NULL){
            //intializing newnode as head
            head = createNode(data);
            p = head;
        }
        else{
        p->next = createNode(data);
        p = p->next;
        }
    }
    //return the list of 5nodes
    return head;
}
//return position of the search element
int search(node *head,int search){
     int pos=1;
     while(head != NULL){
          if(head->data == search) return pos;
          head = head->next;
          pos++;
     }
     //not found condition
     return -1;
}
//main
int main(int argc, char const *argv[])
{
     //driver code
     node *head = createList();
     int s;
     printf("element to be searched\n");
     scanf("%d",&s);
     printf("found at %d\n",search(head,s));

     return 0;
}
```

Output:



Conclusion:

The code works for all cases, and gives expected output. The time complexity of search is same that of a linear search, i.e O(n) for worst, best, average cases. The sapce complexity is constant.

PROGRAM 43:

Title: Implementation of Linked Lists

Objective:

Write a C program to perform the following tasks:

1. Insert a node at beginning of a singly-linked list.
2. Insert a node at end of a singly-linked list.
3. Insert a node at middle of a singly-linked list.
4. Delete a node from the beginning of the singly-linked list.
5. Delete a node from the end of a singly-linked list.

Explanation:

Insertion in beggining of a list: When we add a new node the new node becomes the head of the list thus this function needs to return the head of the newnode
Insertion in end of a list: This function needs to add a new node at a=end of node and if the list is emoty return a newnode.
Insertion in middle of a list: This function needs to add a node in n/2th postion where n is the total nubmer of nodes in the linked-list.
Deletion in beginning of a list: This function needs to delete the free the memory of first node and return the the next node of the head. If empty should return empty.
Deletion in end of a list: This function needs to delete a node at end if only one node is present return NULL.

Pseudo Code:

>Insertion in the beggining of a list

```
/*This algorithm deals with insertion in the beginning of the linked
list*/
/*Initialize the pointer*/
   1. newptr = CREATENODE(data)//*refer to 41st algo for CREATENODE()
      /*error handling
      IF newptr == NULL
            PRINT “no space”
            RETURN*/
   2. newnode->next = head
   3. RETURN newnode
   4. END
```

>Insertion in the middle of a list

```
1. ptr = head
2. IF ptr == NULL RETURN
3. WHILE ptr->next != NULL
   {
4.    IF head->next->next != NULL
      {
5.      head = head->next->next//this pointer moves faster
6.      ptr = head->next//this pointer moves slower
      }
7. ELSE BREAK
   }
8. tmp = ptr->next
9. ptr->next = CREATENODE(data)
10.  ptr->next->next = tmp
11.  END
```

>Insertion at end of list - recursive method

```
/*Initialize the pointer*/
1. INSERT-END(head,data):
2.    IF head == NULL
3.       RETURN createNode(data)
4.    head->next = INSERT-END(head->next,data)
5. END
```

>Deletion at beginning of list

```
1. IF head == NULL
2.    RETURN head
3. /*free head memory*/
4. RETURN head->next
5. END
```

>Deletion at end of list - recursive method

```
1. DELETE-END(head):
2.    IF head == NULL
3.       RETURN NULL
4.    if(head->next == NULL)
5.       RETURN NULL /*free head memory*/
6.    head->next = DELETE-END(head->next)
7. END
```

Code:

```c
#include <stdio.h>
#include <stdlib.h>
#define init() ((struct node*)malloc(sizeof(struct node)))

typedef struct node
{
     int data;
     struct node *next;
     struct node *prev;
}node;

//function which creates nodes
node* createNode(int data){
    node *n = ((node*)malloc(sizeof(node)));
    n->data = data;
    n->next = NULL;
    return n;
}

//inserts a node in the begging
node* insertBeg(node *head,int data){
     node *newNode = createNode(data);
     newNode->next = head;
     return newNode;

}

//inserts at middle
//if nodes = even it adds at nodes/2
//if nodes = odd adds at nodes/2 + 1
void insertMiddle(node *head,int data){

     node *ptr = head;
     if(ptr == NULL){
          //if list empty doesnot add eleemet
          printf("empty\n");
          return;
     }
     //runns till the end
     while(head->next != NULL){
          if(head->next->next != NULL){
```

```
                head = head->next->next;//runs fast
                ptr = ptr->next;//runs half the iterations
            }
            else{
                break;
            }
        }
        node *temp = ptr->next;
        ptr->next = createNode(data);
        ptr->next->next = temp;

    }

    //recusive function which insert the node at the end
    node* insertEnd(node *head, int data){
        //if empty return a newlist with one element
        if(head == NULL){
            return createNode(data);
        }
        //calls insertend function
        head->next = insertEnd(head->next, data);
        return head;

    }

    //deletes node in the begging
    node* deleteBeg(node *head){
        node *temp = head;
        //node empty thus returns null
        if(head == NULL){
            printf("Empty\n");
            return NULL;
        }
        //ideltes the node
        printf("%d deleted\n",temp->data);
        free(temp);
        //returns the  head's next elements
        return head->next;

    }

    //deletes the last last node(not recursive)
    node* deleteEnd(node *head){
        node *ptr = head;
        //if empty list return NULL
```

```
     if(head == NULL ){
          printf("empty\n");
          return NULL;
     }
     //if a single element returns NULL

    if(head->next == NULL){
          printf("%d deleted\n",head->data);;
          free(temp);
          return NULL;
     }
     //last node deleted
     head->next = deleteEnd(head->next);
     return head;
}

//displays the list
void display(node *head){
    while(head!=NULL){
        printf("%d->",head->data);
        head = head->next;
    }
    printf("NULL\n");
}
//main
int main () {
     int choice,data;
     node *head ;
         while(1){
          //menu
             printf("\n***Main Menu*\n");
             printf("\nChoose one option from the following list
...\n");

printf("\n================================================\n");
             printf("\n1.Insert in begining\n2.Insert at
last\n3.Insert middle.\n4.Delete num at the begining \n5.Delete num at
the end\n6.Display\n7.Exit\n");
             printf("\nEnter your choice?\n");
             scanf("\n%d",&choice);

             //performs operation according to the choice
             switch(choice){

                 case 1:{
```

```
                    printf("Enter the data to be inserted\n");
                    scanf("%d",&data);
                    head = insertBeg(head, data);
                    break;
                }
                case 2:{
                    printf("Enter the data to be inserted\n");
                    scanf("%d",&data);
                    head = insertEnd(head, data);
                    break;
                }

                case 3:{

                    printf("Enter the data to be inserted\n");
                    scanf("%d",&data);
                    insertMiddle(head, data);
                    break;
                }

                case 4:{
                    head = deleteBeg(head);
                    break;
                }

                case 5:{
                    head = deleteEnd(head);
                    break;
                }
                    case 6:{
                          printf("The list:\n");
                          display(head);
                          break;
                    }
                case 7: {exit(0);break;}
            }
        }

    return 0;
}
```

Output:

Add begin, display

```
***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
1
Enter the data to be inserted
21

***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
6
The list:
21->NULL
```

Add End, display

```
***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
2
Enter the data to be inserted
23

***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
6
The list:
```

Add middle, Display

```
***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
4
21 deleted

***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
6
The list:
22->23->NULL
```

Delete beginning, Display

```
***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
4
21 deleted

***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
6
The list:
22->23->NULL
```

Delete end Display

```
***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
5
23 deleted

***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
6
The list:
22->NULL
```

terminate

```
***Main Menu*

Choose one option from the following list ...

==============================================

1.Insert in begining
2.Insert at last
3.Insert middle.
4.Delete num at the begining
5.Delete num at the end
6.Display
7.Exit

Enter your choice?
7

sravImac in ~/Desktop/sindhu/dsa-lab/linked-lists
$
```

Conclusion:

The complexity of insertion in begging and deletion in the begining are constant. While the complexities of deletion at end , insertion at end and insertion in middle are of O(n).

*__Note:__ Insertion-middle explanation: Since we dont usually keep track of the length of the linked list, for figuring out middle, we first need to find the length of the list. lets consider it as n. We can find n by traversing throught whole list till the NULL pointer. And once again travel till n/2 to find the middle node and add node there. This would take a time complexity of O(2N). So instead we can take two node, one node will traverse throught the linked list 2X times faster than other. Thus when the second pointer reaches the end of the list the forst pointer will only tracel through half the list.

PROGRAM 44:

Title: Creation of Doubly Linked list

Objective:

Write a C program to create a doubly linked list with 5 nodes.

Explanation:

Doubly linked list is a type of linked list in which each node apart from storing its data has two links. The first link points to the previous node in the list and the second link points to the next node in the list. The first node of the list has its previous link pointing to NULL similarly the last node of the list has its next node pointing to NULL.

Pseudo Code:

>Create node function

```
/*Assume that it is a function where data is passed*/
1. SET data
2. newnode = NEW node //allocates memory to node
3. node->data = data
4. node->next = NULL
5. node ->prev = NULL
6. RETURN node
7. END
```

>Creating linked-list of 5 nodes

```
1. SET ptr=head=NULL
2. FOR i from 0 to 5
   /*for creating n nodes: i from 0 to n*/
   {
3.    READ data
4.    IF head == NULL: head =  CREATENODE(data); ptr = head
5.    ELSE
      {
6.        temp = CREATENODE(data)
7.        ptr->next = temp
8.        temp->prev = ptr
9.        ptr = ptr->next
      }
   }
10.  RETURN head
11.  END
```

>Displaying linked list

```
   /*head is a instance of the actual head here*/
1. WHILE head != NULL
2. {
3.    PRINT head->data
4.    head = head->next
5. }
6. END
```

Code:

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

//double linked list node
typedef struct node
{
     int data;
     struct node *next;
     struct node *prev;
}node;

node* createNode(int data){
     node *newNode = ((node*)malloc(sizeof(node)));
     newNode->data = data;
     newNode->next = NULL;
     newNode->prev = NULL;
     return newNode;
}
//function which creates a double list of 5 nodes
node* createList(){

    int n=5,data;
    node  *p, *head = NULL,*temp;

    //runs loop 5 times
    while(n--){
        printf("Enter a number\n");
        scanf("%d",&data);
        //for the first node
        if(head == NULL){
            //intializing newnode as head
            head = createNode(data);
            p = head;
        }
```

```
            //fpr other nodes
            else{
                temp = createNode(data);
                p->next = temp;
                temp->prev = p;
                p = p->next;
            }
        }
        //return the list of 5nodes
        return head;
    }

    //displays nodes from head to the end
    void display(node *head){
        while(head != NULL){
            printf("%d ",head->data);
            head = head->next;
        }
        printf("\n");
    }//main
    int main(){
        node *head = createList();
        display(head);
        return 0;
    }
```

Output:

```
b in ~/Desktop/sindhu/Dsa-l
$ ./a.out
Enter a number
21
Enter a number
33
Enter a number
44
Enter a number
55
Enter a number
6
21 33 44 55 6
```

Conclusion:

The code works as expected and the time complexity is similar to that of linked-list. But it occupies more space since it stores bith the address of previous and the next node.

PROGRAM 45:

Title: Creation of Circular Linked list

Objective:

Write a C program to create a circular linked list with 5 nodes.

Explanation:

Circular Linked List is a variation of Linked list in which the first element points to the last element and the last element points to the first element. So the only change in creation function from linked list, is that the nth node instead of pointing to NULL. It points to head.

Pseudo Code:

*create node function is same as 41st problem
>Creating Circular linked-list of 5 nodes

```
1. SET ptr=head=NULL
2. FOR i from 0 to 5
   /*for creating n nodes: i from 0 to n*/
   {
3.    READ data
4.    IF head == NULL: head = ptr = CREATENODE(data)
5.    ELSE
      {
6.        ptr->next = CREATENODE(data)
7.        ptr = ptr->next
      }
   }
8. ptr->next = head//last created node points to head
9. RETURN head
10.  END
```

>Displaying Circular linked list

```
   /*head is a instance of the actual head here*/
1. ptr = head
   //runs till it again reaches the address of where it started
2. WHILE head != ptr
3. {
4.    PRINT head->data
5.    head = head->next
6. }
7. END
```

Code:

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

//declaration of a node
typedef struct Node{
     int data;
     struct Node *next;

}node;

//function which creates nodes
node* createNode(int data){
     node *n = ((node*)malloc(sizeof(node)));
     n->data = data;
     n->next = NULL;
     return n;
}

//function which creates a circualar list of 5 nodes
node* createList(){

     int n=5,data;
     node  *p, *head = NULL;

     //runs loop 5 times
     while(n--){
     printf("Enter a number\n");
     scanf("%d",&data);
     if(head == NULL){
          //intializing newnode as head
          head = createNode(data);
          p = head;
     }
     else{
     p->next = createNode(data);
     p = p->next;
     }
     }
     //intializes the last eleemt next to head
     p->next = head;
     //return the list of 5nodes
     return head;

}
```

```c
//displays elemets in linked list till reaches the head
void display(node *head){
     node *ptr = head;
     //if list is not empty
     if(head != NULL){
     while(head->next != ptr){
          printf("%d->",head->data);
          head = head->next;
     }
     printf("%d connected to %d",head->data,ptr->data);
     }
     //if list is empty
     else
     printf("NULL\n");
}

//main
int main()
{
     //driver code
     node *head=createList();
     display(head);
     return 0;
}
```

Output:

```
b in ~/Desktop/sindhu/Dsa-lab/linked-l
$ ./a.out
Enter a number
6
Enter a number
8
Enter a number
9
Enter a number
6
Enter a number
5
6->8->9->6->5 connected to 6
```

Conclusion:

This code is also a simple variation with linked list. Thus its time complexities are also similar with linked list for insertion and deletion.

PROGRAM 46:

Title: Implementation Stack Using Linked Lists

Objective:

Write a C program to implement the stack using linked lists

Explanation:

The push operator is similar to the insertion in the begging in linked list. An dthe pop operator similar to the seletion from end of the linked list(refer problem 43)

Pseudo Code:

>Pushing in Linked-Stack

```
/*Get new node for the ITEM to be pusshed*/
1. newptr = NEW Node
2. newptr->data = ITEM
3. newptr->next = NULL
   /*Add new node at the top*/
4. IF top == NULL THEN
5.    top = newptr
6. ELSE
   {
7.    newptr->next = top
8.    top = newptr
   }
9. END
```

>Popping from a Linked-Stack

```
1. IF top == NULL THEN
2.    PRINT “Stack Empty, Underflow”
3. ELSE
   {
4.    PRINT top->data
5.    top = top->link
   }
6. END
```

Code:

```
#include <stdio.h>
```

```c
#include <stdlib.h>

//declaration of a node
typedef struct Node{
    int data;
    struct Node *next;

}node;
//function which creates nodes
node* createNode(int data){
    node *n = ((node*)malloc(sizeof(node)));
    n->data = data;
    n->next = NULL;
    return n;
}
//pushes a node into the stack
node* push(node *head,int data){
     node *newNode = createNode(data);
     newNode->next = head;
     return newNode;
}
//pops an element from the stack
node* pop(node *head){
     node *temp = head;
     //node empty thus returns null
     if(head == NULL){
          printf("Empty\n");
          return NULL;
     }
     //deletes the node
     printf("%d deleted\n",temp->data);
     free(temp);
     //returns the stack after popping
     return head->next;
}
//prints the top element
void peek(node *head){
     if(head == NULL) {
          printf("empty\n");
          return;
     }
     printf("%d\n",head->data);
}
//displays the stack from top to the end
void display(node *head){
```

```c
        while(head!=NULL){
            printf("%d\n",head->data);
            head = head->next;
        }
    }
    int main(){
         //intialized variables needed
         node *top = NULL;
         int choice,data;
         //runs loop till user chooses exit --> 5
         while(1){
              //menu
              printf("\n1. Push an element on to the STACK.\n"
                        "2. Pop and element from the STACK.\n"
                        "3. Peek the STACK.\n"
                        "4. Display the STACK.\n"
                        "5. Exit the program.\n");
              scanf("%d",&choice);
              //performs action according the choice
              switch(choice){
                   case 1:{
                        printf("\nEnter an element to add\n");
                        scanf("%d",&data);
                        top = push(top,data);
                        break;
                   }
                   case 2:{
                        top = pop(top);
                        break;
                   }
                   case 3:{
                        peek(top);
                        break;
                   }
                   case 4:{
                        display(top);
                        break;
                   }
                   case 5:{
                        exit(0);
                        break;
                   }
                   default: printf("no such option choose again\n");
              }
         }
```

```
        return 0;
    }
```

Output:

push2 elements

```
1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.
1

Enter an element to add
23

1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.
1

Enter an element to add
22
```

Display

```
1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.
3
22

1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.
4
22
23
```

pop and Display:

```
1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.
2
22 deleted

1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.
4
23
```

Exiting program

```
1. Push an element on to the STACK.
2. Pop and element from the STACK.
3. Peek the STACK.
4. Display the STACK.
5. Exit the program.
5
```

Conclusion:

The code works as expected. Time complexities Pushing is O(1) and popping is O(n), that is the poppping operation is slower compared to the implementation using arrays.

PROGRAM 47:

Title: Implementing queues using LInked lists

Objective:

Write a C program to implement the queue using linked list.

Explanation:

The enqueue is similar to insertion in begging and dequeue is similar to deletion in beginning in linked list. SInce we have refernce point of where to delete in both cases.

Pseudo Code:

>EnQueue in Queue

```
/*Allocate space for ITEM to be inseted*/
1. newptr = NEW node//allcoating memory
2. newptr->info = item; newptr->link = NULL
   /*insert in queue*/
3. IF rear = NULL
   {
4.    front = newptr
5.    rear = newptr
   }
6. ELSE
   {
7.    rear->link = newptr
8.    rear = newptr
   }
9. END
```

>DeQueue in Queue

```
1. IF front = NULL THEN
2.     PRINT “queue empty”
3. ELSE
   {
4.     item = front->info
5.     IF front = rear THEN
6.         front = rear = NULL
7.     ELSE front = ront->link
   }
8. End
```

Code:

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

```c
//declaration of a node
typedef struct Node{
    int data;
    struct Node *next;

}node;

struct Queue {
     node *front, *rear;
};

//function which creates nodes
node* createNode(int data){
    node *n = ((node*)malloc(sizeof(node)));
    n->data = data;
    n->next = NULL;
    return n;
}
void enQueue(struct Queue* q, int data)
{
    // Create a new LL node
    node* temp = createNode(data);

    if (q->rear == NULL) {
        q->front = q->rear = temp;
        return;
    }

    q->rear->next = temp;
    q->rear = temp;
}
void deQueue(struct Queue* q)
{
    if (q->front == NULL)
        return;

    node* temp = q->front;

    q->front = q->front->next;

    if (q->front == NULL)
        q->rear = NULL;

    free(temp);
}
```

```c
void display(struct Queue *q){
     node *save = q->front;
     while(q->front != q->rear){
          printf("%d-> ",q->front->data);
          q->front = q->front->next;
     }
     printf("%d\n",q->rear->data );
     q->front = save;
}
int main(int argc, char const *argv[])
{
     //intialized variables needed
    struct Queue *q = ((struct  Queue*)malloc(sizeof(struct Queue)));
     int choice,data;

     //runs loop till user chooses exit --> 5
     while(1){
          //menu
          printf("\n1. EnQueue an element on to the STACK.\n"
                    "2. Dequeue and element from the STACK.\n"
                    "3. Display the STACK.\n"
                    "4. Exit the program.\n");
          scanf("%d",&choice);
          //performs action according the choice
          switch(choice){
               case 1:{
                    printf("\nEnter an element to add\n");
                    scanf("%d",&data);
                    enQueue(q, data);
                    break;
               }
               case 2:{
                    deQueue(q);
                    break;
               }
               case 3:{
                    display(q);
                    break;
               }

               case 4:{
                    exit(0);
                    break;
               }
               default: printf("no such option choose again\n");
```

```
                }
            }
            return 0;
        }
```

Output:

Enqueue- 2 elements

```
1. EnQueue an element on to the STACK.
2. Dequeue and element from the STACK.
3. Display the STACK.
4. Exit the program.
1

Enter an element to add
21

1. EnQueue an element on to the STACK.
2. Dequeue and element from the STACK.
3. Display the STACK.
4. Exit the program.
1

Enter an element to add
22
```

Display

```
1. EnQueue an element on to the STACK.
2. Dequeue and element from the STACK.
3. Display the STACK.
4. Exit the program.
3
21-> 22
```

dequeue and Display:

```
1. EnQueue an element on to the STACK.
2. Dequeue and element from the STACK.
3. Display the STACK.
4. Exit the program.
2

1. EnQueue an element on to the STACK.
2. Dequeue and element from the STACK.
3. Display the STACK.
4. Exit the program.
3
22
```

Exiting program

```
1. EnQueue an element on to the STACK.
2. Dequeue and element from the STACK.
3. Display the STACK.
4. Exit the program.
4
```

Conclusion:

The time complexity of both insertion and seleetion is constant since we have refernce address where to delete. To display the time complexity is O(n).