TITLE : LIST

OBJECTIVE

→ To get familiar with the list (Linked list)

→ To implement the concept of linked list

THEORY

A linked list is a collection of elements called 'nodes' where each node consists of two parts :

- Info : Actual element to be stored in the list. It is called data field.
- Next : One or more link that points to next and previous node in the list. It is also called pointer field.

The link list is a dynamic structure i.e it grows or shrinks depending on different operations performed. The whole list is pointed to by an external pointer called head which contains the address of the first node. It is not the part of linked list. The last node has some specified valve called NULL as next address which means the end of list.

Types of Linked List

- Single linked list
- Doubly linked list
- Circular Linked list
- Doubly Circular linked list.

Single Linked List

It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same datatype. The node contains a pointer to next node means that the node stores the address of the next node in the sequence. A single linked list allows traversal of data only in one way.

```
struct node
{
    int data;
    struct node * next;
};
```

ALGORITHM 1 (INSERTION AT BEGINNING)

1) Create a node using malloc function.
   newnode = (struct node *) malloc (sizeof (struct node));
2) Assign data to info field of new node.
   newnode -> info = data;
3) if head is NULL then set head = newnode and exit
   head = newnode;
4) otherwise
   i) set next of newnode to head
      newnode -> next = head;
   ii) Set the head pointer to point to the new node
      head = newnode;
5) End

ALGORITHM 2 (INSERTION AT END)

1) Create a node using malloc function.
   newnode = (struct node *) malloc (sizeof (struct node));
2) Assign data to info field of new node
   newnode -> info = data;
3) Set next of newnode to NULL
   newnode -> next = NULL;
4) if head is NULL then set head = newnode and exit
   head = newnode;
5) otherwise
   i) Set
      ptr = head;
   ii) Find the last node
      while (ptr -> next != NULL)
      {
         ptr = ptr -> next;
      }
   iii) Set ptr -> next = newnode
      ptr -> next = newnode;
6) End.

ALGORITHM 3 (INSERTION AT SPECIFIC POSITION)

1) Create a node using malloc function
   newnode = (struct node *) malloc (sizeof(struct node));
2) Assign data to info field of newnode
   newnode -> info = data;
3) Enter the position of a node at which you want to insert a newnode
4) Let the position be pos.
5) Set
```
ptr = head;
for (i=0; i<pos-1; i++)
{ ptr = ptr->next;
  if (ptr==NULL)
  {
      printf("Position not found \n");
  }   return;
}
```
6) Set : newnode -> next = ptr -> next;
7) Set the next of ptr to point to the newnode
   ptr -> next = newnode;
8) End

ALGORITHM 4 (DELETION AT BEGINNING)

1) if (head == NULL) then print void deletion and exit i.e.
```
if (ptr==NULL)
{
    printf("List is Empty :\n");
    return;
}
```
2) Otherwise store the address of the first node in temporary variable ptr
   ptr = head;
3) Set head of the next node to head
   head = head -> next;
4) Free the memory reserved by temp variable
   free (ptr);
5) End.

ALGORITHM 5 (DELETION AT THE END OF LIST)

1) If (head == NULL) then print void deletion and exit i.e.
   if (ptr == NULL)
   { printf ("List is empty\n");
     return;
   }
2) Otherwise if (head -> next == NULL) then set ptr = head, head = NULL and free ptr (ie)
   elseif (head -> next == NULL)
   { ptr = head;
     head = NULL;
     printf("The deleted element is %d \t", ptr -> info);
     free (ptr);
   }
3) Otherwise
   ptr = head;
   while (ptr -> next != NULL)
   { temp = ptr;
     ptr = ptr -> next;
   }
   temp -> next = NULL;
   printf ("The deleted element is : %d \t", ptr -> info);
   free (ptr);
4) End.

ALGORITHM 6 (DELETION OF SPECIFIED NODE)

1) If head = NULL, print empty list and exit i.e.

```
if (head == NULL)
{
    printf("The list is empty\n");
    exit(0);
}
```

2) Otherwise

P) Enter the position pos of the node to be deleted.

ii) If pos = 0

(x) Set ptr = head and head = head -> next and free ptr i.e.

```
ptr = head;
head = head -> next;
printf("The deleted element is %d\t", ptr -> info);
free(ptr);
```

iii) Otherwise,

(x) Set

```
ptr = head;
for (i=0; i<pos; i++)
{
    temp = ptr;
    ptr = ptr -> next;
    if (ptr == NULL)
    {
        printf("\n Position not found");
        return;
    }
}
```

(xx) Set

```
temp -> next = ptr -> next;
```

(xxx) Free ptr ie

```
free(ptr);
```

3) End

SOURCE CODE

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

struct node
{
    int info;
    struct node *link;
};
struct node *start = NULL;

void createList()
{
  if (start == NULL)
  {
      int i,n;
      printf("Enter the number of nodes :\n");
      scanf("%d",&n);
      if (n != 0)
      {
          int data;
          struct node *newnode;
          struct node *temp;
          newnode = malloc(sizeof(struct node));
          start = newnode;
          temp = start;
          printf("Enter number to be inserted :\n");
          scanf("%d",&data);
          start->info = data;
          for (i=2;i<=n;i++)
          {
              newnode = malloc(sizeof(struct node));
              temp->link = newnode;
```

```
                printf("Enter number to be inserted \n");
                scanf("%d",&data);
                newnode->info=data;
                temp=temp->link;
            }
            printf("The list is created \n");
        }
        else
        {
            printf("The list is already created \n");
        }
}
void traverse()
{
    struct node *temp;
    if(start==NULL)
    {
        printf("List is empty \n");
    }
    else
    {
        temp=start;
        while(temp!=NULL)
        {
            printf("%d \t",temp->info);
            temp=temp->link;
        }
    }
}
void ~~insertAtfront~~ insertAtFront()
{
    int data;
    struct node *temp;
    temp=malloc(sizeof(struct node));
    printf("Enter the number to be inserted \n");
    scanf("%d",&data);
```

```
    temp->info = data;
    temp->link = start;
    start = temp;
}
void insertAtEnd()
{
    int data;
    struct node *temp, *head;
    temp = malloc(sizeof(struct node));
    printf("Enter number to be inserted: \n");
    scanf("%d", &data);
    temp->link = 0;
    temp->info = data;
    head = start;
    while(head->link != NULL)
    {
        head = head->link;
    }
    head->link = temp;
}
void insertAtPosition()
{
    struct node *temp, *newnode;
    int pos, data, i = 1;
    newnode = malloc(sizeof(struct node));
    printf("Enter Position and data: \n");
    scanf("%d %d", &pos, &data);
    temp = start;
    newnode->info = data;
    newnode->link = 0;
    while(i<pos-1)
    {
        temp = temp->link;
        i++;
    }
```

```
        newnode -> link = temp -> link;
        temp -> link = newnode;
}
void deleteFirst()
{
        struct node * temp;
        if (start == NULL)
        {
                printf("List is empty\n");
        }
        else
        {
                temp = start;
                start = start -> link;
                free(temp);
        }
}
void deleteEnd()
{
        struct node *temp, *prevnode;
        if (start == NULL)
        {
                printf("List is Empty\n");
        }
        else
        {
                temp = start;
                while (temp -> link != 0)
                {
                        prevnode = temp;
                        temp = temp -> link;
                }
                free(temp);
                prevnode -> link = 0;
        }
}
void deletePosition()
{
        struct node *temp, *position;
        int i = 1, pos;
```

```
    if(start==NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        printf("Enter index :\n");
        scanf("%d",&pos);
        position=malloc(sizeof(struct node));
        temp=start;
        while(i<pos-1)
        {
            temp=temp->link;
            i++;
        }
        position=temp->link;
        temp->link=position->link;
        free(position);
    }
}
int main()
{
    int choice;
    while(1)
    {
        printf("1 To see list\n");
        printf("2 For insertion at starting\n");
        printf("3 For insertion at end\n");
        printf("4 For insertion at any position\n");
        printf("5 For deletion of first element\n");
        printf("6 For deletion of last element\n");
        printf("7 For deletion of element at any position\n");
        printf("8 to exit\n");
        printf("Enter choice:\n");
        scanf("%d",&choice);
```

```
switch (choice)
{
        case 1 :
                traverse ();
                break;
        case 2 :
                insertAtFront();
                break;
        case 3 :
                insertAtEnd();
                break;
        case 4 :
                insertAtPosition();
                break;
        case 5 :
                deleteFirst();
                break;
        case 6 :
                deleteEnd();
                break;
        case 7 :
                deleteposition();
                break;
        case 8 :
                exit(1);
        default :
                printf("Incorrect choice\n");
}
return 0;
}
```

DISCUSSION AND CONCLUSION

With the reference to the theoretical knowledge regarding the linked list and the concept of the singly linked list, we were able to write the codes in 'c' language performing the following operations :

- Insertion at first position
- Insertion at last position
- Insertion at Specified position
- Deletion at first position
- Deletion at last position
- Deletion at specified position.

During the above mentioned activities that we performed, we found out how a singly linked list operation mechanism works. The main focus was to implement the singly linked list.

Concluding the lab work, we were able to meet our objective of getting familiar with the list and to implement it. The algorithms mentioned helped us to write the codes. which are mentioned in this lab report with the corresponding output.

OUTPUT

```
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter choice :
1
List is empty

1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position.
8 To exit
Enter choice :
2
Enter number to be inserted : 1

1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position.
8 To exit
Enter choice :
3
Enter number to be inserted : 2
1 To see list.
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position.
8 To exit
```

```
Enter choice:
4
Enter position and data:
3
3
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position.
8 To exit
Enter choice:
1

1 2 3
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position.
8 To exit
Enter choice
5
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter choice
1
2 3
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
```

```
Enter choice :
6
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter choice :
1

2
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter choice :
7
Enter index :
2
1 To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position
8 To exit
Enter choice :
1
List is empty
1 For To see list
2 For insertion at starting
3 For insertion at end
4 For insertion at any position
5 For deletion of first element
6 For deletion of last element
7 For deletion of element at any position.
8 To exit
Enter choice :
8
```