Name : Kshitij Vyas
SE 3 C
Roll No. : 62

Experiment no 6:

Aim: Implementation of Singly Linked List

Objective : It is used to implement stacks and queue which are linked needs throughout computer science .To prevent the Collision between the data in the Hash map.we use a singly Linked list

Theory ;

- Linked List can be defined as collection of objects called **nodes** that are randomly stored in the memory.
- A node contains two fields i.e. data stored at that particular address and the pointer which contains the address of the next node in the memory.
- The last node of the list contains pointer to the null.

Algorithm

```
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3:          Apply Process to PTR->DATA
Step 4:          SET PTR = PTR->NEXT
      [END OF LOOP]
Step 5: EXIT
```

The syntax for creating a node

```
struct Node
{
  int Data;
  Struct Node *next;
};
```

Insertion of a node

```
void insertStart (struct Node **head, int data)
{

  struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));

  newNode - >
  data = data;
  newNode - >
  next = *head;

  //changing the new head to this freshly entered node
  *head = newNode;
}
```

Deletion of a node

```
void deleteStart(struct Node **head)
{
  struct Node *temp = *head;

  // if there are no nodes in Linked List can't delete
  if (*head == NULL)
    {
      printf ("Linked List Empty, nothing to delete");
      return;
    }

  // move head to next node
  *head = (*head)->next;

  free (temp);
```

}

Traversal in a Singly Linked List

```
void display(struct Node* node)
{
   printf("Linked List: ");

   // as linked list will end when Node is Null
   while(node!=NULL){
      printf("%d ",node->data);
      node = node->next;
   }

   printf("\n");
}
```

Code

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *head;

void beginsert ();
void lastinsert ();
void begin_delete();
void last_delete();
void display();
void count();
void main ()
{
   int choice =0;
   while(choice != 9)
   {
      printf("\n1. Insert in begining\n2. Insert at last\n3. Delete from Beginning\n4. Delete from
last\n5. Display\n6. Count\n7. Exit\n");
      printf("\nEnter your choice?\n");
      scanf("\n%d",&choice);
      switch(choice)
      {
```

```
            case 1:
            beginsert();
            break;
            case 2:
            lastinsert();
            break;
            case 3:
            begin_delete();
            break;
            case 4:
            last_delete();
            break;
            case 5:
            display();
            break;
            case 6:
            count();
            break;
            case 7:
            exit(0);
            break;
            default:
            printf("Please enter valid choice..");
        }
    }
}
void beginsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }
```

```
}
void lastinsert()
{
   struct node *ptr,*temp;
   int item;
   ptr = (struct node*)malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW");
   }
   else
   {
      printf("\nEnter value?\n");
      scanf("%d",&item);
      ptr->data = item;
      if(head == NULL)
      {
         ptr -> next = NULL;
         head = ptr;
         printf("\nNode inserted");
      }
      else
      {
         temp = head;
         while (temp -> next != NULL)
         {
            temp = temp -> next;
         }
         temp->next = ptr;
         ptr->next = NULL;
         printf("\nNode inserted");

      }
   }
}

void begin_delete()
{
   struct node *ptr;
   if(head == NULL)
   {
      printf("\nList is empty\n");
   }
```

```
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ...\n");
    }
}

void last_delete()
{
    struct node *ptr,*ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ...\n");
    }

    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ...\n");
    }
}

void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
```

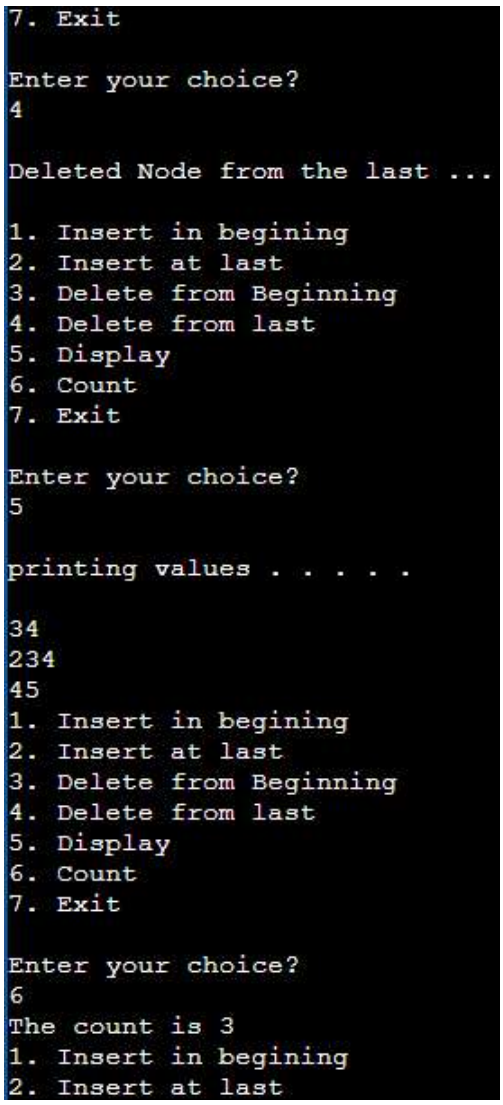
```
    }
    else
    {
        printf("\nprinting values . . . . .\n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
        }
    }
}

void count()
{
    int count=0;
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to count");
    }
    else
    {
        while (ptr!=NULL)
        {
            ptr = ptr -> next;
            count++;
        }
        printf("The count is %d", count);
    }
}
```

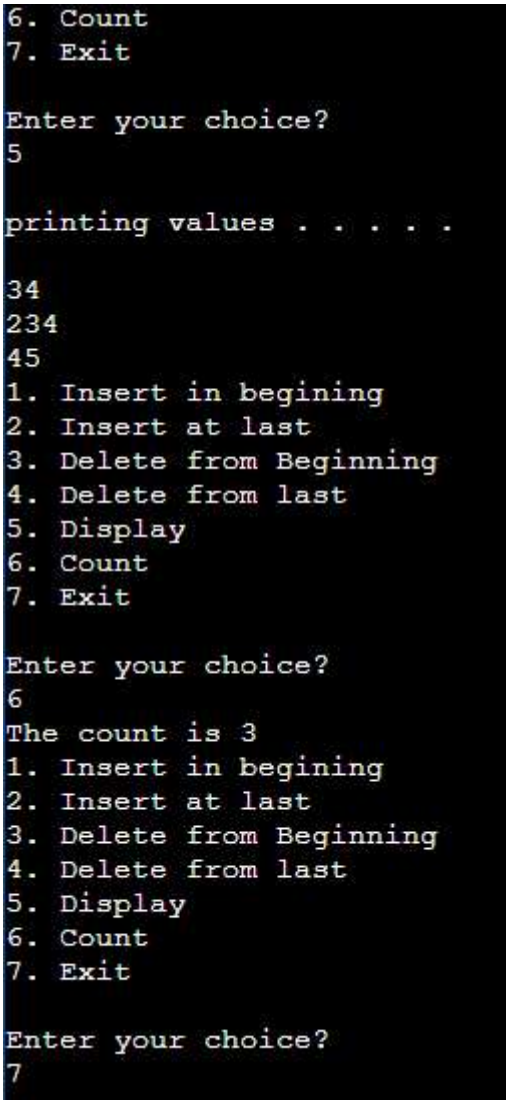
Output

```
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
7. Exit

Enter your choice?
6
Nothing to print
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
7. Exit

Enter your choice?
1

Enter value
45

Node inserted
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
7. Exit

Enter your choice?
```

```
Enter your choice?
2

Enter value?
456

Node inserted
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
7. Exit

Enter your choice?
1

Enter value
234

Node inserted
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
7. Exit

Enter your choice?
1

Enter value
34
```

```
Node inserted
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
7. Exit

Enter your choice?
6
The count is 4
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
7. Exit

Enter your choice?
5

printing values . . . . .

34
234
45
456
1. Insert in begining
2. Insert at last
3. Delete from Beginning
4. Delete from last
5. Display
6. Count
```





Conclusion :

A singly linked list is a type of linked list that is unidirectional, that is, it can be traversed in only one direction from head to the last node (tail). Each element in a linked list is called a node. A single node contains data and a pointer to the next node which helps in maintaining the structure of the list.