Experiment 06

Name : Vaibhav Tatkare
SE Comps 3 C
52

Aim: Implementation of Singly Linked List

Objective : It is used ti 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 :
Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.

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
```

Code :

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

struct node {
  int value;
  struct node *next;
};

void insert();
void display();
void delete();
int count();

typedef struct node DATA_NODE;

DATA_NODE *head_node, *first_node, *temp_node = 0, *prev_node, next_node;
int data;

int main() {
  int option = 0;
  clrscr();

  printf("Singly Linked List Example - All Operations\n");

  while (option < 5) {

    printf("\nOptions\n");
    printf("1 : Insert into Linked List \n");
```

```
    printf("2 : Delete from Linked List \n");
    printf("3 : Display Linked List\n");
    printf("4 : Count Linked List\n");
    printf("Others : Exit()\n");
    printf("Enter your option:");
    scanf("%d", &option);
    switch (option) {
      case 1:
        insert();
        break;
      case 2:
        delete();
        break;
      case 3:
        display();
        break;
      case 4:
        count();
        break;
      default:
        break;
    }
  }

  return 0;
}

void insert() {
  printf("\nEnter Element for Insert Linked List : \n");
  scanf("%d", &data);

  temp_node = (DATA_NODE *) malloc(sizeof (DATA_NODE));

  temp_node->value = data;
```

```
  if (first_node == 0) {
    first_node = temp_node;
  } else {
    head_node->next = temp_node;
  }
  temp_node->next = 0;
  head_node = temp_node;
  fflush(stdin);
}

void delete() {
  int countvalue, pos, i = 0;
  countvalue = count();
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");

  printf("\nEnter Position for Delete Element : \n");
  scanf("%d", &pos);

  if (pos > 0 && pos <= countvalue) {
    if (pos == 1) {
      temp_node = temp_node -> next;
      first_node = temp_node;
      printf("\nDeleted Successfully \n\n");
    } else {
      while (temp_node != 0) {
        if (i == (pos - 1)) {
          prev_node->next = temp_node->next;
          if(i == (countvalue - 1))
          {
        head_node = prev_node;
      }
          printf("\nDeleted Successfully \n\n");
          break;
        } else {
```

```
        i++;
        prev_node = temp_node;
        temp_node = temp_node -> next;
       }
      }
     }
  } else
    printf("\nInvalid Position \n\n");
}

void display() {
  int count = 0;
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");
  while (temp_node != 0) {
    printf("# %d # ", temp_node->value);
    count++;
    temp_node = temp_node -> next;
  }
  printf("\nNo Of Items In Linked List : %d\n", count);
}

int count() {
  int count = 0;
  temp_node = first_node;
  while (temp_node != 0) {
    count++;
    temp_node = temp_node -> next;
  }
  printf("\nNo Of Items In Linked List : %d\n", count);
  getch();
  return count;
}
```

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;
  *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);
}
```

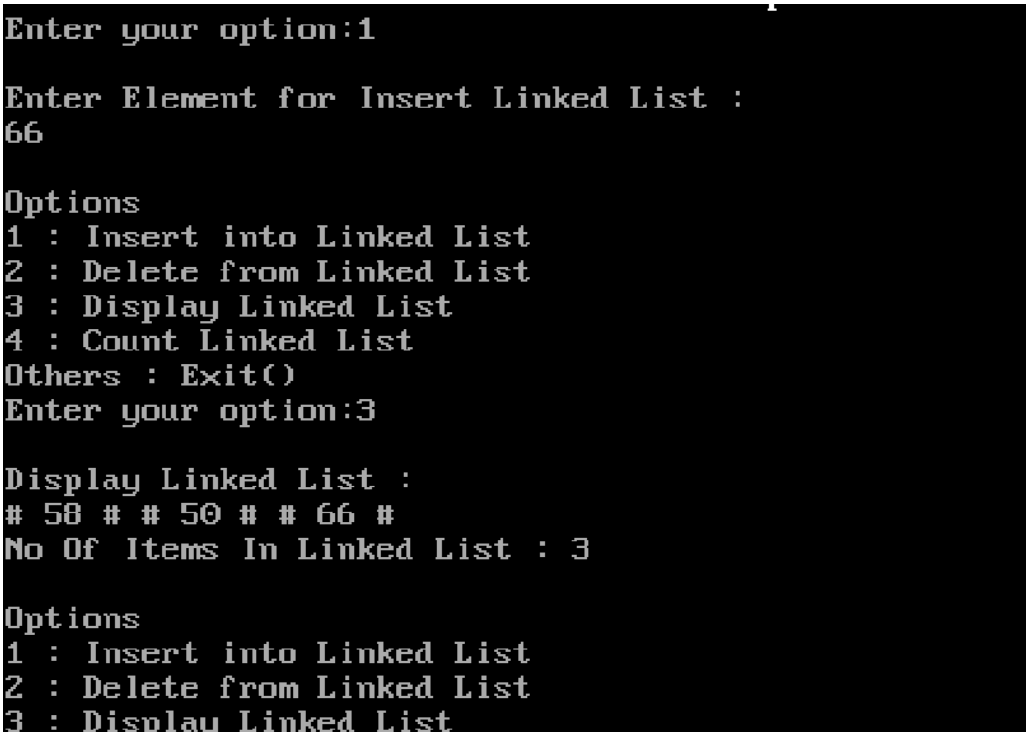
Traversal in a Singly Linked List

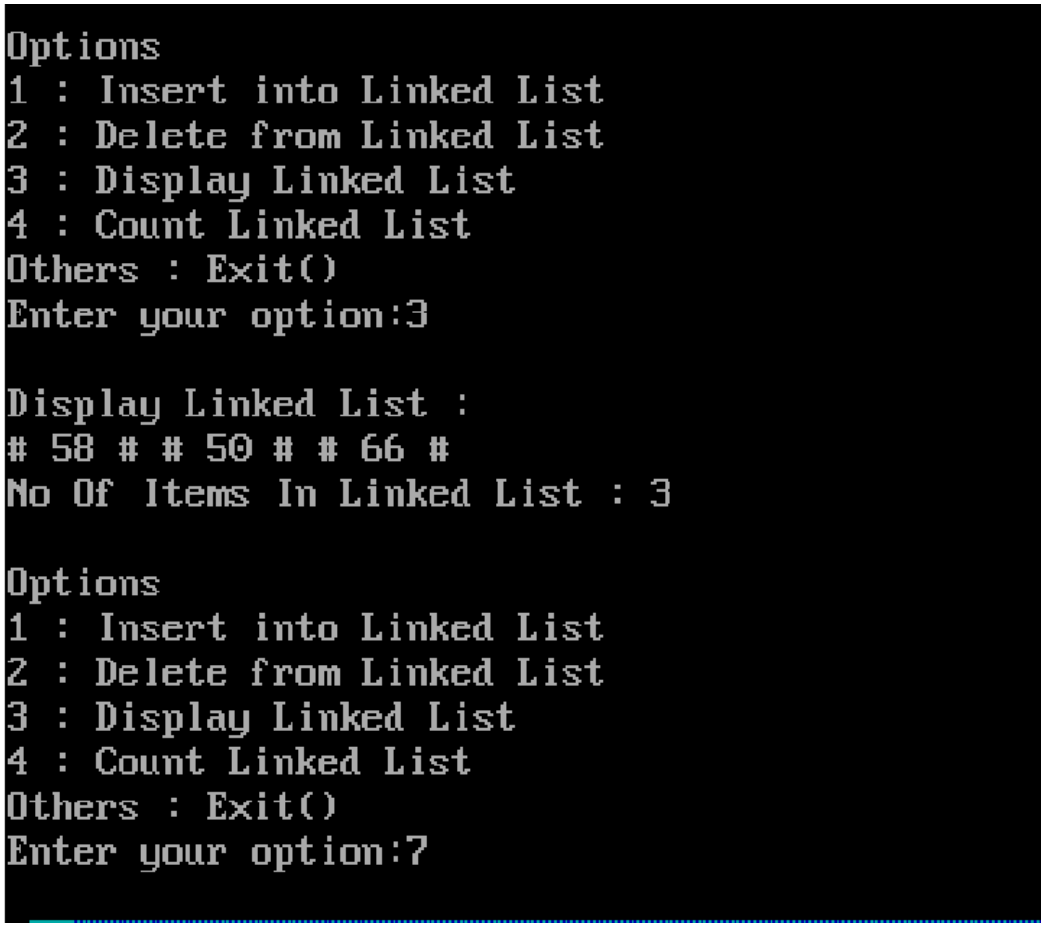
```
void display(struct Node* node)
{
    printf("Linked List: ");

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

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

Output :





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.