NAME: GAURAV KISHOR PATIL

DIV: 2 ROLL NO:54

BATCH: C

Experiment No.6
Implement Singly Linked List ADT
Date of Performance:
Date of Submission:

Experiment No. 6: Singly Linked List Operations

Aim: Implementation of Singly Linked List

Objective:

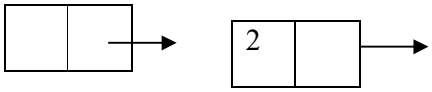
It is used to implement stacks and queues which are like fundamental needs throughout computer science. To prevent the collision between the data in the hash map, we use a singly linked list.

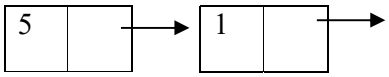
Theory :

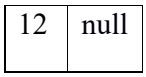
A linked list is a ordered collection of finite, homogenous elements referred as a node. Each node consist of two fields: one field for data which is referred as information and other field is an address field to store the address of next element in the list.

The address field of last node contains null value to indicate the end of list. The elements of linked list are not stored in continuous memory location but they are scattered, and still bounded to each other by an explicit link

The structure of linked list is as shown below







Header is a node containing null in its information field and an next address field contains the address of first data node in the list. Various operations can be performed on singly linked list like insertion at front, end and at specified position , deletion at front, end and at specified position, traversal, copying and merging.

Algorithm

Algorithm : INSERT_SPECIFIED(Header, X, Key)

Input : Header is a pointer to header node. X is a data of node to be inserted and Key is data of node after which insertion is to be done.

Output :A singly linked list enriched with newly inserted node.

Data Structure : A singly linked list whose address of starting node is in Header. And two fields info and next to point to data field and address field respectively. ptr is used to an address of current node

1. new = GETNODE()
2. if new = NULL then
   print "Insufficient Memory"
   Exit
3. else
   ptr = HEADER
   while info(ptr)!= Key AND next(ptr)!= NULL do
   ptr = next(ptr)
   end while
3. if next(ptr) = NULL then
   print "Key not found"
   exit
4. else
   next(new) = next(ptr)
   info(new) = x
   next(ptr) = new
   end if
   end if
5. stop

Algorithm : DELETE_SPECIFIED(Header, Key)

Input : Header is a pointer to header node. Key is data of node after which is to be deleted.

Output :A singly linked list with removed node.

Data Structure : A singly linked list whose address of starting node is in Header. And two

fields info and next to point to data field and address field respectively. ptr is used to an address of current node

1. ptr1 = HEADER
   ptr = next(ptr1)
2. while ptr!= NULL do
   if info(ptr) != Key
      ptr1 = ptr
      ptr = next(ptr)
   else
      next(ptr1) = next(ptr)
      print(info(ptr))
   FREENODE(ptr)
   End if
   End while
3. if ptr = NULL then
   print " key not found"
   end if
4. stop

Algorithm : TRAVERSAL(Header)

Input : Header is a pointer to header node.

Output :A singly linked list is traversed and its data value is printed.

Data Structure : A singly linked list whose address of starting node is in Header. And two fields info and next to point to data field and address field respectively. ptr is used to an address of current node

1. ptr = next(HEADER)
2. while ptr!= NULL do
   print (Info(ptr))
   End while
3. stop

Code:

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

```
#include<stdlib.h>

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

struct node *head = NULL;

void display();
void insert_at_start(int n);
void insert_at_end(int n);
void insert_in_middle(int n);
void delete_at_start();
void delete_at_end();
void deleteFromMid();
void main() {
   int choice, ele, position;

   do {
      printf("Menu Details\n");
      printf("1) Insert at Start\n");
      printf("2) Insert at End\n");
      printf("3) Insert in Middle\n");
      printf("4) Delete at Start\n");
      printf("5) Delete at End\n");
```

```
    printf("6) Delete at Middle\n");
printf("7) Display\n");
printf("8) Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);

switch (choice) {
   case 1:
      printf("Enter the element you want to insert: ");
      scanf("%d", &ele);
      insert_at_start(ele);
      break;
   case 2:
      printf("Enter the element you want to insert: ");
      scanf("%d", &ele);
      insert_at_end(ele);
      break;
   case 3:
      printf("Enter the element you want to insert: ");
      scanf("%d", &ele);
     insert_in_middle(ele);
     break;
   case 4:
      delete_at_start();
      break;
   case 5:
```

```
            delete_at_end();
            break;
        case 6:
            deleteFromMid();
            break;
        case 7:
        display();
            break;
            case 8:printf("Program Closed\n");
            break;

        default:
            printf("Invalid Choice\n");
            break;
      }

   } while (choice != 7);
}

void display() {
   struct node *temp = head;

   if (temp == NULL) {
      printf("Linked List is empty\n");
   } else {
      printf("The Elements are:\n");
```

```
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}

void insert_at_start(int n) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = n;
    newnode->next = head;
    head = newnode;
}

void insert_at_end(int n) {
    struct node *newnode = (struct node*)malloc(sizeof(struct node));
    newnode->data = n;
    newnode->next = NULL;

    if (head == NULL) {
        head = newnode;
        return;
    }

    struct node *temp = head;
```

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

void deleteFromMid() {
    int size;
    struct node *temp, *current,*ct;


    if(head == NULL) {
        printf("List is empty \n");
        return;
    }
    else {
          ct=head;
          while(ct!=NULL){
            ct=ct->next;
            size++;
          }

        int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);


        if( head != NULL ) {
```

```
    temp = head;
    current = NULL;



    for(int i = 0; i < count-1; i++){
        current = temp;
        temp = temp->next;
    }
    if(current != NULL) {

        current->next = temp->next;

        temp = NULL;
    }

    else {
        head = temp->next;

        temp = NULL;
    }
}

else {
    head = NULL;
}
```

```
    }
    size--;
}

void delete_at_start() {
    if (head == NULL) {
        printf("Underflow\n");
    } else {
        struct node *temp = head;
        head = head->next;
        printf("The element deleted: %d\n", temp->data);
        free(temp);
    }
}

void delete_at_end() {
    if (head == NULL) {
        printf("Underflow\n");
    } else if (head->next == NULL) {
        printf("The element deleted: %d\n", head->data);
        free(head);
        head = NULL;
    } else {
        struct node *temp = head;
        while (temp->next->next != NULL) {
            temp = temp->next;
```

```
        }
        printf("The element deleted: %d\n", temp->next->data);
        free(temp->next);
        temp->next = NULL;
    }
}
void insert_in_middle(int n)
{
    struct node *ct;
    int size;

    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = n;
    newNode->next = NULL;


    if(head == NULL) {

        head = newNode;

    }
    else {
         ct=head;
          while(ct!=NULL){
            ct=ct->next;
            size++;
```

```
        }

        struct node *temp, *current;
        //Store the mid position of the list
        int count = (size % 2 == 0) ? (size/2) : ((size+1)/2);

        temp = head;
        current = NULL;


        for(int i = 0; i < count; i++) {

            current = temp;

            temp = temp->next;
        }


        current->next = newNode;

        newNode->next = temp;
    }
    size++;
}
```

Output:

```
3) Insert in Middle
4) Delete at Start
5) Delete at End
6) Delete at Middle
7) Display
8) Exit
Enter your choice: 1
Enter the element you want to insert: 45
Menu Details
1) Insert at Start
2) Insert at End
3) Insert in Middle
4) Delete at Start
5) Delete at End
6) Delete at Middle
7) Display
8) Exit
Enter your choice: 1
Enter the element you want to insert: 56
Menu Details
1) Insert at Start
2) Insert at End
3) Insert in Middle
4) Delete at Start
5) Delete at End
6) Delete at Middle
7) Display
8) Exit
Enter your choice: 1
Enter the element you want to insert: 88
Menu Details
1) Insert at Start
2) Insert at End
3) Insert in Middle
4) Delete at Start
5) Delete at End
6) Delete at Middle
7) Display
8) Exit
Enter your choice: 7
The Elements are:
88 56 45
PS G:\Programs\DS>
```

Conclusion:

A singly linked list in C is a fundamental data structure where each element, known as a node, holds a value and a pointer to the next node in the list. The start of the list is referred to as the head, and the end of the list (which points to NULL) is known as the tail. Singly linked lists are dynamic and allow for efficient insertion and deletion of nodes at any position in the list. However, they do not support direct access to individual elements; one must traverse the list from the head to reach a specific node. Despite this limitation, singly linked lists are

widely used due to their simplicity and flexibility in handling data in scenarios where direct access is not a priority

Time Complexity :0(n(square))

Application:

- Singly linked lists are used to implement other data structures like stack and queue.
- They are used in many real-world applications such as representing polynomials with one or two variables