**Exp 6: Implementation of Singly Linked List**

NAME: TRUSHANT RATHOD

BRANCH: COMPS DIV: 3 ROLL NO: 18

**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 :**

1)Node Structure: A node with data and a pointer to the next node.

2)Creating Nodes: Dynamically allocate memory for nodes and initialize them.

3)Insertion: Add nodes to the list by adjusting pointers.

4)Deletion: Remove nodes by updating pointers and freeing memory.

5)Traversal: Traverse the list using next pointers.

6)Searching: Find elements by comparing data.

7)Displaying: Print data sequentially.

8)Memory Management: Free memory properly.

9)Error Handling: Implement error checks for robustness.

Understanding this theory is essential for data manipulation in programming

**Algorithm:**

### 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\* createNode(int data) {

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

if (newNode == NULL) {

printf("Memory allocation failed.\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

return newNode;

}

void insertEnd(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

} else {

struct Node\* current = \*head;

while (current->next != NULL) {

current = current->next;

}

current->next = newNode;

}

}

void deleteNode(struct Node\*\* head, int key) {

struct Node\* current = \*head;

struct Node\* prev = NULL;

if (current != NULL && current->data == key) {

\*head = current->next;

free(current);

return;

}

while (current != NULL && current->data != key) {

prev = current;

current = current->next;

}

if (current == NULL) {

printf("Node with value %d not found.\n", key);

return;

}

prev->next = current->next;

free(current);

}

void traverseList(struct Node\* head) {

struct Node\* current = head;

printf("Linked List: ");

while (current != NULL) {

printf("%d -> ", current->data);

current = current->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL; //

int n, data, key;

printf("Enter the number of elements to insert: ");

scanf("%d", &n);

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

printf("Enter element %d: ", i + 1);

scanf("%d", &data);

insertEnd(&head, data);

}

traverseList(head);

printf("Enter the value to delete: ");

scanf("%d", &key);

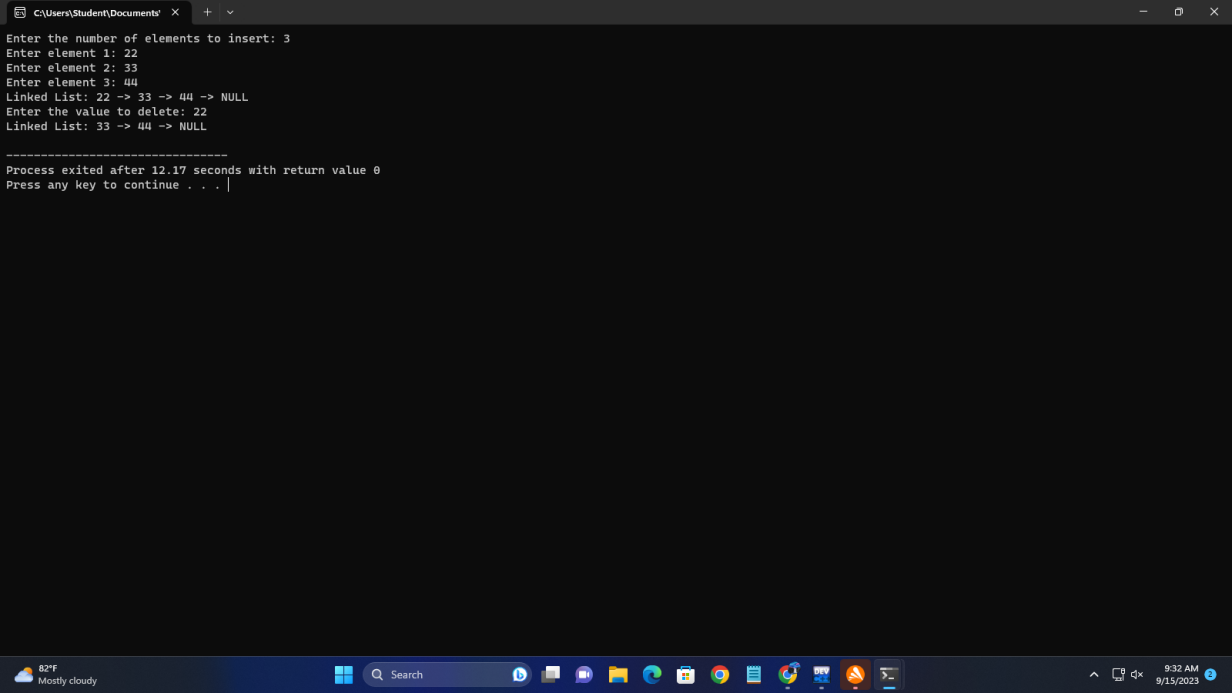
deleteNode(&head, key);

traverseList(head);

return 0;

}

**Output :**



**Conclusion:** In conclusion, the implementation of a Singly Linked List in C provides a flexible and efficient data structure for storing and managing data sequentially. It involves defining a structure for nodes, creating nodes dynamically, and linking them together to form a list. Key operations like insertion, traversal, and display are essential for managing the list.