Experiment no 6:

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

Algorithm:

Define a structure for a singly linked list node (struct Node) with two fields: data and next, representing the data stored in the node and a pointer to the next node.

Implement an insert function that takes the current head of the linked list and data as input. This function allocates memory for a new node, assigns the data, and updates the next pointer to point to the current head. It returns the new head.

Implement a deleteNode function that takes the current head and a key as input. It traverses the linked list, searching for a node with the specified key. If found, it removes the node from the list and frees the memory. It returns the updated head.

Implement a printList function that takes the head as input and prints the elements of the linked list by traversing it.

Implement a quit function that takes the head as input and frees the memory allocated for the entire linked list.

In the main function, create a menu-driven program using a while loop. It repeatedly prompts the user for choices:

Insert: Takes data as input and calls the insert function to add a node to the linked list.

Delete: Takes data as input and calls the deleteNode function to remove a node with the specified data.

Print: Calls the printList function to display the linked list.

Quit: Calls the quit function to free the memory and exits the program.

Handle invalid choices with appropriate messages.

Code:#include <stdio.h>

#include <stdlib.h>

struct Node

{

int data;

struct Node\* next;

};

struct Node\* insert(struct Node\* head, int data)

{

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

newNode->data = data;

newNode->next = head;

return newNode;

}

struct Node\* 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 head;

}

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

prev = current;

current = current->next;

}

if (current == NULL) {

printf("Key not found in the linked list.\n");

return head;

}

prev->next = current->next;

free(current);

return head;

}

void printList(struct Node\* head) {

struct Node\* current = head;

printf("Linked List: ");

while (current != NULL) {

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

current = current->next;

}

printf("NULL\n");

}

void quit(struct Node\* head) {

struct Node\* current = head;

while (current != NULL) {

struct Node\* temp = current;

current = current->next;

free(temp);

}

}

int main() {

struct Node\* head = NULL;

int choice, data;

while (1) {

printf("\nLinked List Operations\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Print\n");

//printf("4. Quiet\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter data to insert: ");

scanf("%d", &data);

head = insert(head, data);

break;

case 2:

printf("Enter data to delete: ");

scanf("%d", &data);

head = deleteNode(head, data);

break;

case 3:

printList(head);

break;

case 4:

quit(head);

exit(0);

default:

printf("Invalid choice, please try again.\n");

}

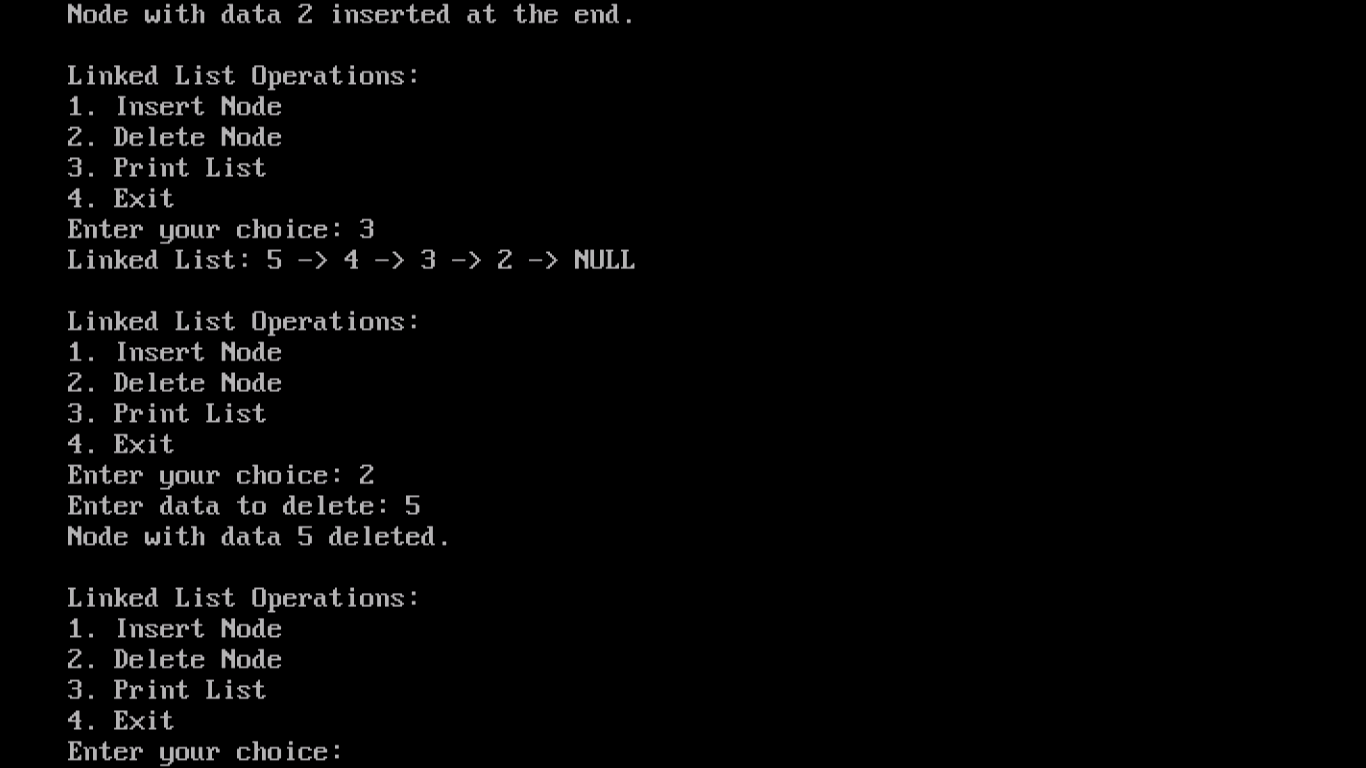
}

return 0;

getch();

}

Output:



Conclusion :

In summary, the provided C code and algorithm offer a basic framework for creating and managing a singly linked list. It includes essential functions for inserting, deleting, printing, and quitting, all accessible through a user-friendly menu-driven interface. This code serves as a foundational example for understanding linked list operations in C, providing a starting point for more advanced applications and demonstrating the versatility of linked lists in data management.