**Batch:A2 Roll No.:16010123032**

**Experiment No. 6**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title:** Implementation of various types of LL- doubly LL, circular LL, circular doubly LL |

**Objective:** To understand the use of linked lists as data structures for various applications.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO 2** | Apply linear and non-linear data structure in application development. |

**Books/ Journals/ Websites referred:**

**Introduction:**

Define Linked List

A **linked list** is a linear data structure where elements, called **nodes**, are connected using **pointers**. Each node contains two components:

1. **Data**: The value or information the node holds.
2. **Pointer**: A reference (or link) to the next node in the sequence.

Unlike arrays, linked lists do not store elements in contiguous memory locations. This dynamic structure allows efficient insertion and deletion operations, but accessing an element requires traversal from the head node. There are several types of linked lists, such as **singly linked lists (SLL)**, **doubly linked lists (DLL)**, and **circular linked lists**.

**Types of linked list:**

 **Singly Linked List (SLL):**

* Each node contains data and a pointer to the next node.
* Traversal is only possible in one direction (from head to tail).

 **Doubly Linked List (DLL):**

* Each node has two pointers: one to the next node and one to the previous node.
* Supports traversal in both directions (forward and backward).

 **Circular Linked List:**

* The last node points back to the first node, forming a circular structure.
* Can be singly or doubly linked.
* Traversal can continue indefinitely unless stopped.

 **Circular Doubly Linked List:**

* A combination of circular and doubly linked lists.
* Each node points to both the next and previous nodes, and the last node points to the first node.

**Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:**

**Program:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

struct Node\* prev;

} Node;

Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

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

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

void deleteNode(Node\*\* head, int value) {

Node\* temp = \*head;

while (temp != NULL) {

if (temp->data == value) {

if (temp->prev != NULL) {

temp->prev->next = temp->next;

} else {

\*head = temp->next;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

free(temp);

printf("Node with value %d deleted.\n", value);

return;

}

temp = temp->next;

}

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

}

Node\* searchNode(Node\* head, int value) {

Node\* temp = head;

while (temp != NULL) {

if (temp->data == value) {

return temp;

}

temp = temp->next;

}

return NULL;

}

void displayForward(Node\* head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node\* temp = head;

while (temp != NULL) {

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

temp = temp->next;

}

printf("\n");

}

void displayBackward(Node\* head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

while (temp != NULL) {

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

temp = temp->prev;

}

printf("\n");

}

void displayMenu() {

printf("\nDoubly Linked List Menu:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Search\n");

printf("4. Display\n");

printf("5. Exit\n");

}

int main() {

Node\* head = NULL;

int choice, value;

do {

displayMenu();

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

insertEnd(&head, value);

printf("Node with value %d inserted.\n", value);

break;

case 2:

printf("Enter value to delete: ");

scanf("%d", &value);

deleteNode(&head, value);

break;

case 3:

printf("Enter value to search: ");

scanf("%d", &value);

Node\* result = searchNode(head, value);

if (result != NULL) {

printf("Node with value %d found.\n", value);

} else {

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

}

break;

case 4:

printf("List in forward direction:\n");

displayForward(head);

printf("List in backward direction:\n");

displayBackward(head);

break;

case 5:

printf("Exiting...\n");

while (head != NULL) {

Node\* temp = head;

head = head->next;

free(temp);

}

break;

default:

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

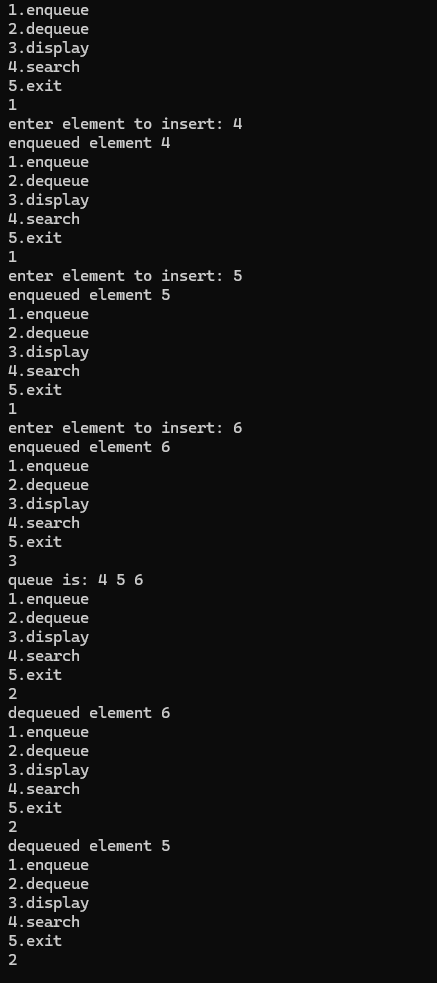
}

} while (choice != 5);

return 0;

}

OUTPUT:



**Circular linked list**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = newNode; // Points to itself initially

return newNode;

}

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

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

Node\* temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = \*head;

}

void deleteNode(Node\*\* head, int value) {

if (\*head == NULL) {

printf("List is empty.\n");

return;

}

Node \*temp = \*head, \*prev = NULL;

do {

if (temp->data == value) {

if (prev == NULL) { // Node to delete is head

Node\* toDelete = \*head;

if (toDelete->next == \*head) { // Only one node in the list

\*head = NULL;

free(toDelete);

return;

}

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = (\*head)->next;

\*head = (\*head)->next;

free(toDelete);

} else { // Node to delete is not head

prev->next = temp->next;

free(temp);

}

printf("Node with value %d deleted.\n", value);

return;

}

prev = temp;

temp = temp->next;

} while (temp != \*head);

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

}

Node\* searchNode(Node\* head, int value) {

Node\* temp = head;

if (head == NULL) return NULL;

do {

if (temp->data == value) {

return temp;

}

temp = temp->next;

} while (temp != head);

return NULL;

}

void displayList(Node\* head) {

if (head == NULL) {

printf("List is empty.\n");

return;

}

Node\* temp = head;

do {

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

temp = temp->next;

} while (temp != head);

printf("\n");

}

void displayMenu() {

printf("\nCircular Linked List Menu:\n");

printf("1. Insert\n");

printf("2. Delete\n");

printf("3. Search\n");

printf("4. Display\n");

printf("5. Exit\n");

}

int main() {

Node\* head = NULL;

int choice, value;

do {

displayMenu();

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

insertEnd(&head, value);

printf("Node with value %d inserted.\n", value);

break;

case 2:

printf("Enter value to delete: ");

scanf("%d", &value);

deleteNode(&head, value);

break;

case 3:

printf("Enter value to search: ");

scanf("%d", &value);

Node\* result = searchNode(head, value);

if (result != NULL) {

printf("Node with value %d found.\n", value);

} else {

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

}

break;

case 4:

printf("List contents:\n");

displayList(head);

break;

case 5:

printf("Exiting...\n");

while (head != NULL) {

Node\* temp = head;

if (head->next == head) { // Only one node in the list

head = NULL;

free(temp);

break;

}

while (temp->next != head) {

temp = temp->next;

}

temp->next = head->next;

temp = head;

head = head->next;

free(temp);

}

break;

default:

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

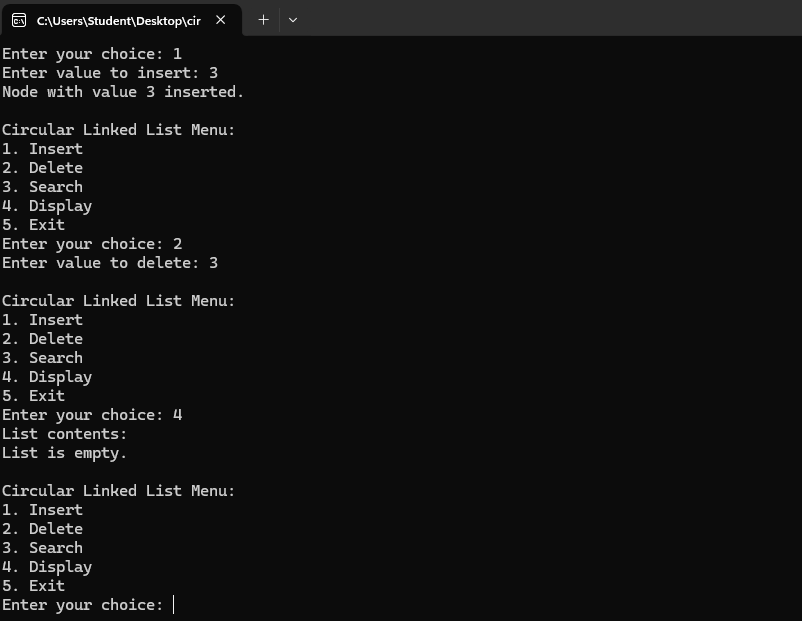
}

} while (choice != 5);

return 0;

}

**Output:-**



**Conclusion:-**

**Post lab questions:**

1. Compare and contrast SLL and DLL

 **Structure:**

* SLL has a single pointer to the next node.
* DLL has pointers to both the next and previous nodes.

 **Memory Usage:**

* SLL uses less memory as it only stores one pointer per node.
* DLL uses more memory with two pointers per node.

 **Traversal:**

* SLL can only be traversed forward.
* DLL allows bidirectional traversal.

 **Insertion/Deletion:**

* SLL is simpler but requires traversing from the head for deletion or insertion.
* DLL allows easier and faster insertions/deletions, especially in the middle.

 **Complexity:**

* SLL is simpler to implement.
* DLL is more complex due to maintaining two pointers per node.

1. Priority Queue

* **Element Priority:**
  + Each element in a priority queue has a priority level. Elements with higher priority are dequeued before those with lower priority.
* **Ordering:**
  + Unlike a regular queue, which follows FIFO (First In, First Out), a priority queue serves elements based on priority, not arrival time.
* **Heap Implementation:**
  + Priority queues are often implemented using a heap (usually a binary heap) to ensure efficient insertion and removal operations.
* **Time Complexity:**
  + Insertion and deletion of the highest-priority element take **O(log n)** time in a heap-based priority queue, making it efficient for large datasets.
* **Applications:**
  + Commonly used in algorithms like Dijkstra’s shortest path, Huffman coding, and for task scheduling in operating systems.

Top of Form

Bottom of Form