Experiment 6:

Represent a node of Doubly linked linear list and implement the following functions.

1) Create a list

2) insert an element – at the beginning, at the end and at a specified position in the list

3) delete an element from the beginning, end or a specified position at the list

4) reverse the list

5) search for an element in the list.

Also create a menu-driven program to test all the functions.

C:

//Experiment 6 DSALAB III SEMESTER

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* next;

    struct Node\* prev;

};

struct Node\* head = NULL;

void createList() {

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

    head->data = 0;

    head->next = NULL;

    head->prev = NULL;

    printf("List created successfully!\n");

}

void insertAtBeginning(int value) {

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

    newNode->data = value;

    newNode->next = head;

    newNode->prev = NULL;

    head->prev = newNode;

    head = newNode;

    printf("Element inserted at the beginning successfully!\n");

}

void insertAtEnd(int value) {

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

    newNode->data = value;

    newNode->next = NULL;

    newNode->prev = NULL;

    struct Node\* temp = head;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = newNode;

    newNode->prev = temp;

    printf("Element inserted at the end successfully!\n");

}

void insertAtPosition(int value, int position) {

    if (position < 1) {

        printf("Invalid position!\n");

        return;

    }

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

    newNode->data = value;

    if (position == 1) {

        newNode->next = head;

        newNode->prev = NULL;

        head->prev = newNode;

        head = newNode;

        printf("Element inserted at position %d successfully!\n", position);

        return;

    }

    struct Node\* temp = head;

    for (int i = 1; i < position-1; i++) {

        if (temp->next == NULL) {

            printf("Invalid position!\n");

            return;

        }

        temp = temp->next;

    }

    newNode->next = temp->next;

    newNode->prev = temp;

    temp->next->prev = newNode;

    temp->next = newNode;

    printf("Element inserted at position %d successfully!\n", position);

}

void deleteFromBeginning() {

    if (head == NULL) {

        printf("List is empty!\n");

        return;

    }

    struct Node\* temp = head;

    head = head->next;

    head->prev = NULL;

    free(temp);

    printf("Element deleted from the beginning successfully!\n");

}

void deleteFromEnd() {

    if (head == NULL) {

        printf("List is empty!\n");

        return;

    }

    if (head->next == NULL) {

        free(head);

        head = NULL;

        printf("Element deleted from the end successfully!\n");

        return;

    }

    struct Node\* temp = head;

    while (temp->next->next != NULL) {

        temp = temp->next;

    }

    struct Node\* temp2 = temp->next;

    temp->next = NULL;

    free(temp2);

    printf("Element deleted from the end successfully!\n");

}

void deleteFromPosition(int position) {

    if (head == NULL) {

        printf("List is empty!\n");

        return;

    }

    if (position < 1) {

        printf("Invalid position!\n");

        return;

    }

    if (position == 1) {

        struct Node\* temp = head;

        head = head->next;

        head->prev = NULL;

        free(temp);

        printf("Element deleted from position %d successfully!\n", position);

        return;

    }

    struct Node\* temp = head;

    for (int i = 1; i < position-1; i++) {

        if (temp->next == NULL) {

            printf("Invalid position!\n");

            return;

        }

        temp = temp->next;

    }

    struct Node\* temp2 = temp->next;

    temp->next = temp->next->next;

    temp->next->prev = temp;

    free(temp2);

    printf("Element deleted from position %d successfully!\n", position);

}

int searchElement(int value) {

    if (head == NULL) {

        printf("List is empty!\n");

        return -1;

    }

    struct Node\* temp = head;

    int position = 1;

    while (temp != NULL) {

        if (temp->data == value) {

            printf("Element found at position %d\n", position);

            return position;

        }

        temp = temp->next;

        position++;

    }

    printf("Element not found in the list!\n");

    return -1;

}

void reverseList() {

    if (head == NULL) {

        printf("List is empty!\n");

        return;

    }

    struct Node\* temp = head;

    struct Node\* nextNode;

    struct Node\* prevNode = NULL;

    while (temp != NULL) {

        nextNode = temp->next;

        temp->next = prevNode;

        temp->prev = nextNode;

        prevNode = temp;

        temp = nextNode;

    }

    head = prevNode;

    printf("List reversed successfully!\n");

}

void displayList() {

    if (head == NULL) {

        printf("List is empty!\n");

        return;

    }

    struct Node\* temp = head;

    printf("List: ");

    while (temp != NULL) {

        printf("%d ", temp->data);

        temp = temp->next;

    }

    printf("\n");

}

void sortList() {

    if (head == NULL) {

        printf("List is empty!\n");

        return;

    }

    struct Node\* temp1 = head;

    while (temp1->next != NULL) {

        struct Node\* temp2 = temp1->next;

        while (temp2 != NULL) {

            if (temp1->data > temp2->data) {

                int temp = temp1->data;

                temp1->data = temp2->data;

                temp2->data = temp;

            }

            temp2 = temp2->next;

        }

        temp1 = temp1->next;

    }

    printf("List sorted successfully!\n");

}

int main() {

    int choice, value, position;

    while (1) {

        printf("Menu\n");

        printf("1. Create List\n");

        printf("2. Insert at Beginning\n");

        printf("3. Insert at End\n");

        printf("4. Insert at Position\n");

        printf("5. Delete from Beginning\n");

        printf("6. Delete from End\n");

        printf("7. Delete from Position\n");

        printf("8. Search Element\n");

        printf("9. Reverse List\n");

        printf("10. Sort List\n");

        printf("11. Display List\n");

        printf("12. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                createList();

                break;

            case 2:

                printf("Enter value to insert: ");

                scanf("%d", &value);

                insertAtBeginning(value);

                break;

            case 3:

                printf("Enter value to insert: ");

                scanf("%d", &value);

                insertAtEnd(value);

                break;

            case 4:

                printf("Enter value to insert: ");

                scanf("%d", &value);

                printf("Enter position to insert at: ");

                scanf("%d", &position);

                insertAtPosition(value, position);

                break;

            case 5:

                deleteFromBeginning();

                break;

            case 6:

                deleteFromEnd();

                break;

            case 7:

                printf("Enter position to delete from: ");

                scanf("%d", &position);

                deleteFromPosition(position);

                break;

            case 8:

                printf("Enter value to search for: ");

                scanf("%d", &value);

                searchElement(value);

                break;

            case 9:

                reverseList();

                break;

            case 10:

                sortList();

                break;

            case 11:

                displayList();

                break;

                case 12:

                exit(0);

                break;

            default:

                printf("Invalid choice!\n");

                break;

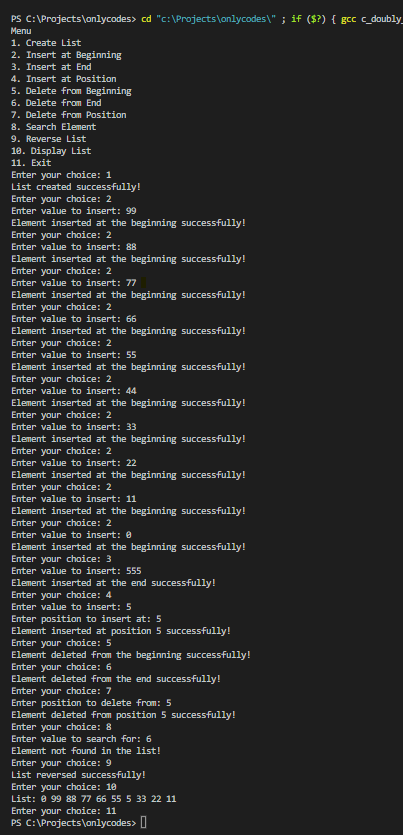
        }

    }

    return 0;

}

Output:



Test Cases:

|  |  |  |
| --- | --- | --- |
| Input | Expected Output | Actual Output |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) displayList() | 5 8 10 | 5 8 10 |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) deleteFromBeginning() displayList() | 8 10 | 8 10 |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) deleteFromEnd() displayList() | 5 8 | 5 8 |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) deleteFromPosition(2) displayList() | 5 10 | 5 10 |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) reverseList() displayList() | 10 8 5 | 10 8 5 |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) searchElement(8) | Found at position 2 | Found at position 2 |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) searchElement(15) | Not Found | Not Found |
| createList() insertAtBeginning(5) insertAtEnd(10) insertAtPosition(8, 2) sortList() displayList() | 5 8 10 | 5 8 10 |

|  |  |  |
| --- | --- | --- |
| createList()  insertAtEnd(5) insertAtEnd(10) insertAtEnd(8)  reverseList()  displayList() | 8 10 5 | 8 10 5 |
| createList()  insertAtEnd(5) insertAtEnd(10) insertAtEnd(8)  sortList()  displayList() | 5 8 10 | 5 8 10 |