#include <stdio.h>

#include <stdlib.h>

// Structure for a node

struct Node {

int data;

struct Node\* next;

};

struct Node\* head = NULL; // Global head pointer

// Function to create a new node

struct Node\* createNode(int value) {

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

newNode->data = value;

newNode->next = NULL;

return newNode;

}

// Insertion at the beginning

void insertAtBeginning(int value) {

struct Node\* newNode = createNode(value);

newNode->next = head;

head = newNode;

printf("Inserted %d at the beginning.\n", value);

}

// Insertion at the end

void insertAtEnd(int value) {

struct Node\* newNode = createNode(value);

if (head == NULL) {

head = newNode;

} else {

struct Node\* temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

}

printf("Inserted %d at the end.\n", value);

}

// Insertion at any position

void insertAtPosition(int value, int pos) {

struct Node\* newNode = createNode(value);

if (pos == 1) {

newNode->next = head;

head = newNode;

printf("Inserted %d at position %d.\n", value, pos);

return;

}

struct Node\* temp = head;

for (int i = 1; i < pos - 1 && temp != NULL; i++) {

temp = temp->next;

}

if (temp == NULL) {

printf("Position %d is invalid.\n", pos);

free(newNode);

return;

}

newNode->next = temp->next;

temp->next = newNode;

printf("Inserted %d at position %d.\n", value, pos);

}

// Delete from the beginning

void deleteAtBeginning() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

head = head->next;

printf("Deleted %d from the beginning.\n", temp->data);

free(temp);

}

// Delete from the end

void deleteAtEnd() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

struct Node\* prev = NULL;

while (temp->next != NULL) {

prev = temp;

temp = temp->next;

}

if (prev == NULL)

head = NULL;

else

prev->next = NULL;

printf("Deleted %d from the end.\n", temp->data);

free(temp);

}

// Delete from any position

void deleteAtPosition(int pos) {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

if (pos == 1) {

head = head->next;

printf("Deleted %d from position %d.\n", temp->data, pos);

free(temp);

return;

}

struct Node\* prev = NULL;

for (int i = 1; i < pos && temp != NULL; i++) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) {

printf("Position %d is invalid.\n", pos);

return;

}

prev->next = temp->next;

printf("Deleted %d from position %d.\n", temp->data, pos);

free(temp);

}

// Display the linked list

void display() {

if (head == NULL) {

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

return;

}

struct Node\* temp = head;

printf("Linked List elements: ");

while (temp != NULL) {

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

temp = temp->next;

}

printf("NULL\n");

}

// Main function

int main() {

int choice, value, pos;

while (1) {

printf("\n--- Linked List Operations ---\n");

printf("1. Insert at Beginning\n");

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

printf("3. Insert at Position\n");

printf("4. Delete at Beginning\n");

printf("5. Delete at End\n");

printf("6. Delete at Position\n");

printf("7. Display\n");

printf("8. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter value: ");

scanf("%d", &value);

insertAtBeginning(value);

break;

case 2:

printf("Enter value: ");

scanf("%d", &value);

insertAtEnd(value);

break;

case 3:

printf("Enter value and position: ");

scanf("%d %d", &value, &pos);

insertAtPosition(value, pos);

break;

case 4:

deleteAtBeginning();

break;

case 5:

deleteAtEnd();

break;

case 6:

printf("Enter position to delete: ");

scanf("%d", &pos);

deleteAtPosition(pos);

break;

case 7:

display();

break;

case 8:

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

exit(0);

default:

printf("Invalid choice! Try again.\n");

}

}

return 0;

}



