**Topic: Linked List**

1) Write a Menu driven C program to accomplish the following functionalities in single linked list.

a) Create a single linked list. b) Display the elements of a single linked list.

c) Insert a node at the beginning of a single linked list.

d) Insert a node at the end of a single linked list.

e) Insert a node before a given node of a single linked list.

f) Insert a node after a given node of a single linked list.

g) Delete a node from the beginning of a single linked list.

h) Delete a node from the end of a single linked list.

i) Delete a node after a given node of a single linked list.

j) Delete the entire single linked list.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* next;

};

struct Node\* head = NULL;

// Function to create a single linked list

void createList(int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    if (head == NULL) {

        head = newNode;

    } else {

        struct Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

}

// Function to display elements of a single linked list

void displayList() {

    struct Node\* temp = head;

    if (temp == NULL) {

        printf("List is empty.\n");

        return;

    }

    while (temp != NULL) {

        printf("%d -> ", temp->data);

        temp = temp->next;

    }

    printf("NULL\n");

}

// Function to insert a node at the beginning

void insertAtBeginning(int data) {

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

    newNode->data = data;

    newNode->next = head;

    head = newNode;

}

// Function to insert a node at the end

void insertAtEnd(int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    if (head == NULL) {

        head = newNode;

        return;

    }

    struct Node\* temp = head;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = newNode;

}

// Function to insert a node before a given node

void insertBeforeNode(int target, int data) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

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

    newNode->data = data;

    if (head->data == target) {

        newNode->next = head;

        head = newNode;

        return;

    }

    struct Node\* temp = head;

    while (temp->next != NULL && temp->next->data != target) {

        temp = temp->next;

    }

    if (temp->next == NULL) {

        printf("Node not found.\n");

    } else {

        newNode->next = temp->next;

        temp->next = newNode;

    }

}

// Function to insert a node after a given node

void insertAfterNode(int target, int data) {

    struct Node\* temp = head;

    while (temp != NULL && temp->data != target) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Node not found.\n");

        return;

    }

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

    newNode->data = data;

    newNode->next = temp->next;

    temp->next = newNode;

}

// Function to delete a node from the beginning

void deleteFromBeginning() {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = head;

    head = head->next;

    free(temp);

}

// Function to delete a node from the end

void deleteFromEnd() {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    if (head->next == NULL) {

        free(head);

        head = NULL;

        return;

    }

    struct Node\* temp = head;

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

        temp = temp->next;

    }

    free(temp->next);

    temp->next = NULL;

}

// Function to delete a node after a given node

void deleteAfterNode(int target) {

    struct Node\* temp = head;

    while (temp != NULL && temp->data != target) {

        temp = temp->next;

    }

    if (temp == NULL || temp->next == NULL) {

        printf("Node not found or no node exists after the given node.\n");

        return;

    }

    struct Node\* nodeToDelete = temp->next;

    temp->next = temp->next->next;

    free(nodeToDelete);

}

// Function to delete the entire list

void deleteList() {

    struct Node\* temp;

    while (head != NULL) {

        temp = head;

        head = head->next;

        free(temp);

    }

    printf("Entire list deleted.\n");

}

// Main function with menu

int main() {

    int choice, data, target;

    while (1) {

        printf("\nMenu:\n");

        printf("1. Create a single linked list\n");

        printf("2. Display the elements\n");

        printf("3. Insert at the beginning\n");

        printf("4. Insert at the end\n");

        printf("5. Insert before a given node\n");

        printf("6. Insert after a given node\n");

        printf("7. Delete from the beginning\n");

        printf("8. Delete from the end\n");

        printf("9. Delete after a given node\n");

        printf("10. Delete the entire list\n");

        printf("11. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter data to insert: ");

                scanf("%d", &data);

                createList(data);

                break;

            case 2:

                displayList();

                break;

            case 3:

                printf("Enter data to insert at the beginning: ");

                scanf("%d", &data);

                insertAtBeginning(data);

                break;

            case 4:

                printf("Enter data to insert at the end: ");

                scanf("%d", &data);

                insertAtEnd(data);

                break;

            case 5:

                printf("Enter the target node data before which to insert: ");

                scanf("%d", &target);

                printf("Enter data to insert: ");

                scanf("%d", &data);

                insertBeforeNode(target, data);

                break;

            case 6:

                printf("Enter the target node data after which to insert: ");

                scanf("%d", &target);

                printf("Enter data to insert: ");

                scanf("%d", &data);

                insertAfterNode(target, data);

                break;

            case 7:

                deleteFromBeginning();

                break;

            case 8:

                deleteFromEnd();

                break;

            case 9:

                printf("Enter the target node data after which to delete: ");

                scanf("%d", &target);

                deleteAfterNode(target);

                break;

            case 10:

                deleteList();

                break;

            case 11: exit(0);

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

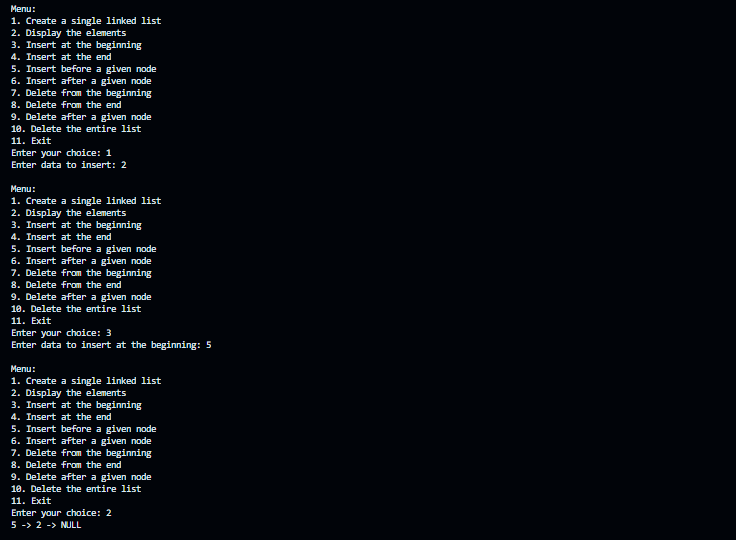
        }

    }

    return 0;

}

**Output:**



2) Write a Menu driven C program to accomplish the following functionalities in circular linked list.

a) Create a circular linked list.

b) Display the elements of a circular linked list.

c) Insert a node at the beginning of a circular linked list.

d) Insert a node at the end of a circular linked list.

e) Delete a node from the beginning of a circular linked list.

f) Delete a node from the end of a circular linked list.

g) Delete a node after a given node of a circular linked list.

h) Delete the entire circular linked list.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the circular linked list

struct Node {

  int data;

  struct Node \*next;

};

// Function to create a circular linked list with a single node

struct Node\* createNode(int data) {

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

  newNode->data = data;

  newNode->next = newNode;

  return newNode;

}

// Function to display all elements in the circular linked list

void display(struct Node\* last) {

  if (last == NULL) {

    printf("List is empty.\n");

    return;

  }

  struct Node\* temp = last->next;

  do {

    printf("%d -> ", temp->data);

    temp = temp->next;

  } while (temp != last->next);

  printf("\n");

}

// Function to insert a node at the beginning of the circular linked list

struct Node\* insertAtBeginning(struct Node\* last, int data) {

  struct Node\* newNode = createNode(data);

  if (last == NULL) {

    last = newNode;

  } else {

    newNode->next = last->next;

    last->next = newNode;

  }

  return last;

}

// Function to insert a node at the end of the circular linked list

struct Node\* insertAtEnd(struct Node\* last, int data) {

  struct Node\* newNode = createNode(data);

  if (last == NULL) {

    return newNode;

  }

  newNode->next = last->next;

  last->next = newNode;

  last = newNode;

  return last;

}

// Function to delete a node from the beginning of the circular linked list

struct Node\* deleteFromBeginning(struct Node\* last) {

  if (last == NULL) {

    printf("List is empty.\n");

    return NULL;

  }

  struct Node\* temp = last->next;

  if (last == temp) {

    free(temp);

    return NULL;

  }

  last->next = temp->next;

  free(temp);

  return last;

}

// Function to delete a node from the end of the circular linked list

struct Node\* deleteFromEnd(struct Node\* last) {

  if (last == NULL) {

    printf("List is empty.\n");

    return NULL;

  }

  struct Node\* temp = last->next;

  if (last == temp) {

    free(last);

    return NULL;

  }

  while (temp->next != last) {

    temp = temp->next;

  }

  temp->next = last->next;

  free(last);

  last = temp;

  return last;

}

// Function to delete a node after a given node in the circular linked list

struct Node\* deleteAfterNode(struct Node\* last, int value) {

  if (last == NULL) {

    printf("List is empty.\n");

    return NULL;

  }

  struct Node\* temp = last->next;

  do {

    if (temp->data == value) {

      struct Node\* nodeToDelete = temp->next;

      if (nodeToDelete == last) {

        last = temp;

      }

      temp->next = nodeToDelete->next;

      free(nodeToDelete);

      return last;

    }

    temp = temp->next;

  } while (temp != last->next);

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

    return last;

}

// Function to delete the entire circular linked list

struct Node\* deleteList(struct Node\* last) {

  if (last == NULL) return NULL;

  struct Node\* current = last->next;

  while (current != last) {

    struct Node\* temp = current;

    current = current->next;

    free(temp);

  }

  free(last);

  printf("Entire list deleted.\n");

  return NULL;

}

int main() {

  struct Node\* last = NULL;

  int choice, data, value;

  do {

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

    printf("1. Create circular linked list\n");

    printf("2. Display elements\n");

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

    printf("4. Insert at end\n");

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

    printf("6. Delete from end\n");

    printf("7. Delete after a node\n");

    printf("8. Delete entire list\n");

    printf("9. Exit\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    switch (choice) {

      case 1:

        printf("Enter data to create list: ");

        scanf("%d", &data);

        last = createNode(data);

        break;

      case 2:

        display(last);

        break;

      case 3:

        printf("Enter data to insert at beginning: ");

        scanf("%d", &data);

        last = insertAtBeginning(last, data);

        break;

      case 4:

        printf("Enter data to insert at end: ");

        scanf("%d", &data);

        last = insertAtEnd(last, data);

        break;

      case 5:

        last = deleteFromBeginning(last);

        break;

      case 6:

        last = deleteFromEnd(last);

        break;

      case 7:

        printf("Enter value after which to delete: ");

        scanf("%d", &value);

        last = deleteAfterNode(last, value);

        break;

      case 8:

        last = deleteList(last);

        break;

      case 9:

        printf("Exiting program.\n");

        break;

      default:

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

    }

  } while (choice != 9);

  return 0;

}

**Output:**

