LINKED- 6

Sunday, 23 March 2025 10:22 PM

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>  /* For getch() and clrscr() in Turbo C++ */

// Structure to represent a node in the linked list
typedef struct Node {
   int data;
   struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (newNode == NULL) {
      printf("Memory allocation failed!\n");
      getch();  /* Wait for key press before exiting */
      exit(1);
   }
   newNode->data = data;
   newNode->next = NULL;
   return newNode;
}

// Function to display the elements of the linked list
void display(Node* head) {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   printf("Linked List elements: ");
   Node* temp = head;
   while (temp != NULL) {
      printf("%d ", temp->data);
      temp = temp->next;
```

```
    }
    printf("\n");
}

// Function to insert an element at the beginning of the linked list
Node* insertAtBeginning(Node* head, int data) {
    Node* newNode = createNode(data);
    newNode->next = head;
    printf("Element %d inserted at the beginning.\n", data);
    return newNode;
}

// Function to insert an element at the end of the linked list
Node* insertAtEnd(Node* head, int data) {
    Node* newNode = createNode(data);

    if (head == NULL) {
        printf("Element %d inserted at the end.\n", data);
        return newNode;
    }

    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = newNode;
    printf("Element %d inserted at the end.\n", data);

    return head;
}

// Function to delete an element from the beginning of the linked list
Node* deleteFromBeginning(Node* head) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }

    Node* temp = head;
    head = head->next;
    printf("Element %d deleted from beginning successfully.\n", temp->data);
    free(temp);
    return head;
}

// Function to delete an element from the end of the linked list
Node* deleteFromEnd(Node* head) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }

    if (head->next == NULL) {
        printf("Element %d deleted from end successfully.\n", head->data);
        free(head);
        return NULL;
    }

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

    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }

    prev->next = NULL;
    printf("Element %d deleted from end successfully.\n", temp->data);
    free(temp);
    return head;
}

// Function to delete a given element from the linked list
Node* deleteElement(Node* head, int data) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }

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

    /* If head node itself holds the key to be deleted */
```

```
   /* If head node itself holds the key to be deleted */
   if (temp != NULL && temp->data == data) {
      head = temp->next;
      free(temp);
      printf("Element %d deleted successfully.\n", data);
      return head;
   }

   /* Search for the key to be deleted, keep track of the
     previous node as we need to change prev->next */
   while (temp != NULL && temp->data != data) {
      prev = temp;
      temp = temp->next;
   }

   /* If key was not present in linked list */
   if (temp == NULL) {
      printf("Element %d not found in the list.\n", data);
      return head;
   }

   /* Unlink the node from linked list */
   prev->next = temp->next;
   free(temp);
   printf("Element %d deleted successfully.\n", data);

   return head;
}

// Function to free all memory allocated for the linked list
void freeList(Node* head) {
   Node* temp;

   while (head != NULL) {
      temp = head;
      head = head->next;
      free(temp);
   }
}

// Main function
int main() {
```

```
int main() {
    Node* head = NULL;
    int choice, data;

    do {
        clrscr();  /* Clear the screen - Turbo C++ specific */
        printf("\nLinked List Operations:\n");
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Delete from beginning\n");
        printf("4. Delete from end\n");
        printf("5. Delete a given element\n");
        printf("6. Display\n");
        printf("7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter element to insert at beginning: ");
                scanf("%d", &data);
                head = insertAtBeginning(head, data);
                break;

            case 2:
                printf("Enter element to insert at end: ");
                scanf("%d", &data);
                head = insertAtEnd(head, data);
                break;

            case 3:
                head = deleteFromBeginning(head);
                break;

            case 4:
                head = deleteFromEnd(head);
                break;

            case 5:
                printf("Enter element to delete: ");
                scanf("%d", &data);
                head = deleteElement(head, data);
                break;

            case 6:
                display(head);
                break;

            case 7:
                printf("Freeing memory and exiting...\n");
                freeList(head);
                break;

            default:
                printf("Invalid choice! Please enter a valid option.\n");
        }

        printf("\nPress any key to continue...");
        getch();  /* Wait for user to press a key before continuing */

    } while (choice != 7);

    return 0;
}
```