```
//OM Dattatray Gavande
//Class:-SY CSE A
//Roll No:-CS2145


#include <stdio.h>
#include <stdlib.h>
// Define Node structure
struct Node {
  int data;
  struct Node *next;
};
// Function to create a new node
struct Node *createNode(int data) {
  struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
  if (!newNode) {
     printf("Memory allocation failed!\n");
     exit(1);
  }
  newNode->data = data;
  newNode->next = NULL;
  return newNode;
}
// insert at the beginning
void insertAtFirst(struct Node** head, int data) {
  struct Node* newNode = createNode(data);
  newNode->next = *head;
  *head = newNode;
}
// insert at the end
void insertAtEnd(struct Node** head, int data) {
  struct Node* newNode = createNode(data);
  if (*head == NULL) {
     *head = newNode;
     return;
  }
  struct Node* temp = *head;
  while (temp->next != NULL) {
     temp = temp->next;
  }
  temp->next = newNode;
}
// insert at specific position
void insertAtPosition(struct Node** head, int data, int position) {
  if (position < 0) {
     printf("Invalid position. Position must be non-negative.\n");
     return;
  }


  if (position == 0) {
     insertAtFirst(head, data);
     return;
```

```
  }

  struct Node* newNode = createNode(data);
  struct Node* temp = *head;

  // Traverse to the node *before* the desired position
  for (int i = 0; temp != NULL && i < position - 1; i++) {
     temp = temp->next;
  }

  // If temp is NULL, it means the position is out of range
  // (e.g., trying to insert at index 5 in a list of size 2)
  if (temp == NULL) {
     printf("Position out of range\n");
     free(newNode); // Free the newly created node
     return;
  }

  newNode->next = temp->next;
  temp->next = newNode;
}

// Delete from beginning
void deleteFromFirst(struct Node** head) {
  if (*head == NULL) {
     printf("List is empty. Cannot delete from beginning.\n");
     return;
  }
  struct Node* temp = *head;
  *head = temp->next;
  free(temp);
  printf("First node deleted.\n");
}

// Delete from end
void deleteFromEnd(struct Node** head) {
  if (*head == NULL) {
     printf("List is empty. Cannot delete from end.\n");
     return;
  }

  // Case 1: Only one node in the list
  if ((*head)->next == NULL) {
     free(*head);
     *head = NULL;
     printf("Last node deleted.\n");
     return;
  }
```

```
  // Case 2: More than one node
  struct Node* temp = *head;
  while (temp->next->next != NULL) {
     temp = temp->next;
  }
  free(temp->next);
  temp->next = NULL;
  printf("Last node deleted.\n");
}


// Delete at position
void deleteAtPosition(struct Node** head, int position) {
  if (*head == NULL) {
     printf("List is empty. Cannot delete.\n");
     return;
  }


  if (position < 0) {
     printf("Invalid position. Position must be non-negative.\n");
     return;
  }


  if (position == 0) {
     deleteFromFirst(head);
     return;
  }


  struct Node* temp = *head;

  // Traverse to the node *before* the desired position
  for (int i = 0; temp != NULL && i < position - 1; i++) {
     temp = temp->next;
  }


  // If temp is NULL or temp->next is NULL, the position is out of range
  if (temp == NULL || temp->next == NULL) {
     printf("Position out of range.\n");
     return;
  }

  struct Node* nodeToDelete = temp->next;
  temp->next = temp->next->next;
  free(nodeToDelete);
  printf("Node at position %d deleted.\n", position);
}
```

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


// Free entire list
void freeList(struct Node** head) {
  struct Node* current = *head;
  struct Node* next;
  while (current != NULL) {
     next = current->next; // Store next before freeing current
     free(current);
     current = next;
  }
  *head = NULL; // Set head to NULL after freeing all nodes
  printf("List freed.\n");
}


// Main function
int main() {
  struct Node* head = NULL;
  int choice, data, position;


  do {
     printf("\n==== Singly Linked List Menu ====\n");
     printf("1. Insert at Beginning\n");
     printf("2. Insert at End\n");
     printf("3. Insert at Position\n");
     printf("4. Delete from Beginning\n");
     printf("5. Delete from End\n");
     printf("6. Delete at Position\n");
     printf("7. Display List\n");
     printf("8. Exit\n");
     printf("Enter your choice: ");

     // Check for successful scanf
     if (scanf("%d", &choice) != 1) {
        // Handle non-integer input to prevent infinite loop
        while(getchar() != '\n');
        choice = -1; // Set to invalid choice
```

```
    }


    switch (choice) {
      case 1:
        printf("Enter data: ");
        scanf("%d", &data);
        insertAtFirst(&head, data); // Corrected function name
        break;
      case 2:
        printf("Enter data: ");
        scanf("%d", &data);
        insertAtEnd(&head, data); // Corrected function name
        break;
      case 3:
        printf("Enter data: ");
        scanf("%d", &data);
        printf("Enter position (0-based index): ");
        scanf("%d", &position);
        insertAtPosition(&head, data, position); // Corrected function name
        break;
      case 4:
        deleteFromFirst(&head);
        break;
      case 5:
        deleteFromEnd(&head);
        break;
      case 6:
        printf("Enter position (0-based Index): ");
        scanf("%d", &position);
        deleteAtPosition(&head, position);
        break;
      case 7:
        printList(head);
        break;
      case 8:
        freeList(&head); // Ensure list is freed on exit
        printf("Exiting program.\n");
        break;
      default:
        printf("Invalid choice! Try again.\n");
    }
  } while (choice != 8);
  return 0;
}
```