MOHAMMAD HATIF OSMANI
B.E. ETCE UG2 G2
002210701084

1. **Create a singly connected linked list in C. Write a menu driven program to perform**
   a. **addition at the beginning**
   b. **addition at any given position**
   c. **addition at the end of the list**
   d. **deletion at the beginning**
   e. **deletion at any given position**
   f. **deletion at the end of the list**

   Also create a display function that displays the entire linked list and call this function after every operation listed above.
   The program should keep on displaying the menu to the user and wait for user's choice.
   The user should be able to quit the program with choice 0.

Code:

```
#include <stdio.h>

#include <stdlib.h>



// Node structure

struct Node {

   int data;

   struct Node* next;

};

// LinkedList structure

struct LinkedList {

  struct Node *head;

};



// Function prototypes (Optional)

struct Node *createNode(int data);

void addAtBeginning(struct LinkedList *list, int data);

void addAtPosition(struct LinkedList *list, int data, int position);

void addAtEnd(struct LinkedList *list, int data);
```

```
void deleteAtBeginning(struct LinkedList *list);
void deleteAtPosition(struct LinkedList *list, int position);
void deleteAtEnd(struct LinkedList *list);
void display(struct LinkedList *list);

// Function to create a new Node
struct Node *createNode(int data) {
  struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = NULL;
  return newNode;
}

// Function to initialize an empty LinkedList
void initLinkedList(struct LinkedList *list) { list->head = NULL; }

// Function to free the memory of the linked list
void destroyLinkedList(struct LinkedList *list) {
  struct Node *current = list->head;
  while (current != NULL) {
    struct Node *temp = current;
    current = current->next;
    free(temp);
  }
  list->head = NULL;
```

```
}

// Main function
int main() {
  struct LinkedList list;
  /*
  IMPORTANT

  DO NOT CHANGE THE BELOW CODE.
  WRITE YOUR OWN CODE FOR FUNCTIONS (See comments after the main program).
  WRITE YOUR FUNCTIONS IN SUCH A WAY THAT
  THEY WILL WORK WITH THE BELOW CODE IN MAIN FUNCTION.

  TRY TO GET AN IDEA ABOUT THE FUNCTION PARAMETERS FROM THE FUNCTION CALLS IN
  THE MAIN FUNCTION.

  YOU CAN ALSO TAKE HINTS FROM THE `destroyLinkedList` FUNCTION WRITTEN ABOVE.
  */
  int i, choice, data, position, testCases;
  scanf("%d", &testCases);
  for (i = 0; i < testCases; i++) {
    printf("\n######################\nTEST CASE %d\n######################\n",
          i + 1);
    initLinkedList(&list);
    do {
```

```
printf("\nMenu:\n");
printf("1. Add at beginning\n");
printf("2. Add at any position\n");
printf("3. Add at end\n");
printf("4. Delete at beginning\n");
printf("5. Delete at any position\n");
printf("6. Delete at end\n");
printf("0. Quit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
printf("%d", choice);
switch (choice) {
case 1:
  printf("\nEnter data: ");
  scanf("%d", &data);
  printf("%d", data);
  addAtBeginning(&list, data);
  display(&list);
  break;
case 2:
  printf("\nEnter data: ");
  scanf("%d", &data);
  printf("%d", data);
  printf("\nEnter position: ");
  scanf("%d", &position);
```

```
  printf("%d", position);
  addAtPosition(&list, data, position);
  display(&list);
  break;
 case 3:
  printf("\nEnter data: ");
  scanf("%d", &data);
  printf("%d", data);
  addAtEnd(&list, data);
  display(&list);
  break;
 case 4:
  deleteAtBeginning(&list);
  display(&list);
  break;
 case 5:
  printf("\nEnter position: ");
  scanf("%d", &position);
  printf("%d", position);
  deleteAtPosition(&list, position);
  display(&list);
  break;
 case 6:
  deleteAtEnd(&list);
  display(&list);
```

```
      break;
    case 0:
      printf("\nExiting program.\n");
      destroyLinkedList(&list);
      break;
    default:
      printf("\nInvalid choice. Please try again.\n");
    }
  } while (choice != 0);
 }
 return 0;
}

// Function to add a new node at the beginning of the linked list
void addAtBeginning(struct LinkedList *list, int data) {
 struct Node *newNode = createNode(data);
 newNode->next = list->head;
 list->head = newNode;
}

// Function to add a new node at a specific position in the linked list
void addAtPosition(struct LinkedList *list, int data, int position) {
 if (position < 1) {
  printf("Invalid position\n");
  return;
```

```
  }

  struct Node *newNode = createNode(data);

  if (position == 1) {
    newNode->next = list->head;
    list->head = newNode;
    return;
  }

  struct Node *current = list->head;
  for (int i = 1; i < position - 1 && current != NULL; ++i) {
    current = current->next;
  }

  if (current == NULL) {
    printf("Invalid position\n");
    return;
  }

  newNode->next = current->next;
  current->next = newNode;
}

// Function to add a new node at the end of the linked list
```

```
void addAtEnd(struct LinkedList *list, int data) {
  struct Node *newNode = createNode(data);

  if (list->head == NULL) {
    list->head = newNode;
    return;
  }

  struct Node *current = list->head;
  while (current->next != NULL) {
    current = current->next;
  }

  current->next = newNode;
}

// Function to delete the node at the beginning of the linked list
void deleteAtBeginning(struct LinkedList *list) {
  if (list->head == NULL) {
    printf("List is empty\n");
    return;
  }

  struct Node *temp = list->head;
  list->head = list->head->next;
```

```
  free(temp);
}

// Function to delete the node at a specific position in the linked list
void deleteAtPosition(struct LinkedList *list, int position) {
  if (position < 1 || list->head == NULL) {
    printf("Invalid position\n");
    return;
  }

  struct Node *current = list->head;
  struct Node *previous = NULL;

  if (position == 1) {
    list->head = current->next;
    free(current);
    return;
  }

  for (int i = 1; i < position && current != NULL; ++i) {
    previous = current;
    current = current->next;
  }

  if (current == NULL) {
```

```
    printf("Invalid position\n");
    return;
  }

  previous->next = current->next;
  free(current);
}


// Function to delete the node at the end of the linked list
void deleteAtEnd(struct LinkedList *list) {
  if (list->head == NULL) {
    printf("List is empty\n");
    return;
  }

  if (list->head->next == NULL) {
    free(list->head);
    list->head = NULL;
    return;
  }

  struct Node *current = list->head;
  struct Node *previous = NULL;
```

```
  while (current->next != NULL) {
    previous = current;
    current = current->next;
  }

  previous->next = NULL;
  free(current);
}

// Function to display the linked list
void display(struct LinkedList *list) {
  struct Node *current = list->head;
  printf("Linked List: ");
  while (current != NULL) {
    printf("%d -> ", current->data);
    current = current->next;
  }
  printf("NULL\n");
}
```

Output

######################

TEST CASE 1

######################

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 1

Enter data: 10Linked List: 10 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 20Linked List: 10 -> 20 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 30Linked List: 10 -> 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 2

Enter data: 25

Enter position: 2Linked List: 10 -> 25 -> 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 40Linked List: 10 -> 25 -> 20 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 25 -> 20 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 5

Enter position: 2Linked List: 25 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 6Linked List: 25 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 1

Enter data: 100Linked List: 100 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 200Linked List: 100 -> 200 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 0

Exiting program.

######################

TEST CASE 2

######################

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 1

Enter data: 10Linked List: 10 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 20Linked List: 10 -> 20 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 30Linked List: 10 -> 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 2

Enter data: 25

Enter position: 20Invalid position

Linked List: 10 -> 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 40Linked List: 10 -> 20 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 20 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 5

Enter position: 2Linked List: 20 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 6Linked List: 20 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4List is empty

Linked List: NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 1

Enter data: 100Linked List: 100 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 200Linked List: 100 -> 200 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 0

Exiting program.

######################

TEST CASE 3

######################

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 1

Enter data: 10Linked List: 10 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 20Linked List: 10 -> 20 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 30Linked List: 10 -> 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 2

Enter data: 25

Enter position: 2Linked List: 10 -> 25 -> 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 40Linked List: 10 -> 25 -> 20 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 25 -> 20 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 5

Enter position: 50Invalid position

Linked List: 25 -> 20 -> 30 -> 40 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 6Linked List: 25 -> 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 20 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 1

Enter data: 100Linked List: 100 -> 30 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 200Linked List: 100 -> 30 -> 200 -> NULL

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 0

Exiting program.

2. Create a circular linked list in C with the same functionalities as 1. Only change the display function which should show the head twice, i.e, the head should again appear at the end of the displayed list which is possible in a circular linked list. Make sure there is no duplicate data in the linked list.

Code:

```
#include <stdio.h>

#include <stdlib.h>


// Node structure

struct Node {

   int data;

   struct Node* next;

};

// LinkedList structure

struct LinkedList {

  struct Node *head;

};


// Function prototypes (Optional)

struct Node *createNode(int data);


struct Node *last = NULL;


void addAtBeginning(struct LinkedList *list, int data);

void addAtPosition(struct LinkedList *list, int data, int position);

void addAtEnd(struct LinkedList *list, int data);
```

```
void deleteAtBeginning(struct LinkedList *list);
void deleteAtPosition(struct LinkedList *list, int position);
void deleteAtEnd(struct LinkedList *list);
void display(struct LinkedList *list);

// Function to create a new Node
struct Node *createNode(int data) {
  struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = NULL;
  return newNode;
}

// Function to initialize an empty LinkedList
void initLinkedList(struct LinkedList *list) { list->head = NULL; }

// Function to free the memory of the linked list
void destroyLinkedList(struct LinkedList *list) {
  struct Node *current = list->head;
  while (current != NULL) {
    struct Node *temp = current;
    current = current->next;
    free(temp);
  }
  list->head = NULL;
```

```
}

// Main function
int main() {
  struct LinkedList list;
  /*
  IMPORTANT

  DO NOT CHANGE THE BELOW CODE.
  WRITE YOUR OWN CODE FOR FUNCTIONS (See comments after the main program).
  WRITE YOUR FUNCTIONS IN SUCH A WAY THAT
  THEY WILL WORK WITH THE BELOW CODE IN MAIN FUNCTION.

  TRY TO GET AN IDEA ABOUT THE FUNCTION PARAMETERS FROM THE FUNCTION CALLS IN
  THE MAIN FUNCTION.

  YOU CAN ALSO TAKE HINTS FROM THE `destroyLinkedList` FUNCTION WRITTEN ABOVE.
  */
  int i, choice, data, position, testCases;
  scanf("%d", &testCases);
  for (i = 0; i < testCases; i++) {
    printf("\n######################\nTEST CASE %d\n######################\n",
          i + 1);
    initLinkedList(&list);
    do {
```

```
printf("\nMenu:\n");
printf("1. Add at beginning\n");
printf("2. Add at any position\n");
printf("3. Add at end\n");
printf("4. Delete at beginning\n");
printf("5. Delete at any position\n");
printf("6. Delete at end\n");
printf("0. Quit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
printf("%d", choice);
switch (choice) {
case 1:
  printf("\nEnter data: ");
  scanf("%d", &data);
  printf("%d", data);
  addAtBeginning(&list, data);
  display(&list);
  break;
case 2:
  printf("\nEnter data: ");
  scanf("%d", &data);
  printf("%d", data);
  printf("\nEnter position: ");
  scanf("%d", &position);
```

```
  printf("%d", position);
  addAtPosition(&list, data, position);
  display(&list);
  break;
 case 3:
  printf("\nEnter data: ");
  scanf("%d", &data);
  printf("%d", data);
  addAtEnd(&list, data);
  display(&list);
  break;
 case 4:
  deleteAtBeginning(&list);
  display(&list);
  break;
 case 5:
  printf("\nEnter position: ");
  scanf("%d", &position);
  printf("%d", position);
  deleteAtPosition(&list, position);
  display(&list);
  break;
 case 6:
  deleteAtEnd(&list);
  display(&list);
```

```
        break;
      case 0:
        printf("\nExiting program.\n");
        destroyLinkedList(&list);
        break;
      default:
        printf("\nInvalid choice. Please try again.\n");
      }
    } while (choice != 0);
  }
  return 0;
}

// Function to add a new node at the beginning of the linked list
void addAtBeginning(struct LinkedList *list, int data) {
    struct Node* newNode = createNode(data);
    if (list->head == NULL) {
        list->head = newNode;
        newNode->next = list->head;  // Make it circular
    } else {
        newNode->next = list->head;
        struct Node* current = list->head;
        while (current->next != list->head) {
            current = current->next;
        }
```

```
        current->next = newNode;
        list->head = newNode;
    }
}

// Function to add a new node at a specific position in the linked list
void addAtPosition(struct LinkedList *list, int data, int position) {
    if (position < 1) {
        printf("Invalid position\n");
        return;
    }

    struct Node* newNode = createNode(data);

    if (position == 1) {
        addAtBeginning(list, data);
        return;
    }

    struct Node* current = list->head;
    for (int i = 1; i < position - 1 && current != NULL; ++i) {
        current = current->next;
    }

    if (current == NULL) {
```

```
        printf("Invalid position\n");
        return;
    }

    newNode->next = current->next;
    current->next = newNode;
}

// Function to add a new node at the end of the linked list
void addAtEnd(struct LinkedList *list, int data){
    struct Node* newNode = createNode(data);

    if (list->head == NULL) {
        list->head = newNode;
        newNode->next = list->head;  // Make it circular
    } else {
        struct Node* current = list->head;
        while (current->next != list->head) {
            current = current->next;
        }
        current->next = newNode;
        newNode->next = list->head;
    }
}
```

```
// Function to delete the node at the beginning of the linked list
void deleteAtBeginning(struct LinkedList *list) {
    if (list->head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* temp = list->head;
    if (list->head->next == list->head) {
        list->head = NULL;
    } else {
        struct Node* current = list->head;
        while (current->next != list->head) {
            current = current->next;
        }
        current->next = list->head->next;
        list->head = list->head->next;
    }
    free(temp);
}


// Function to delete the node at a specific position in the linked list
void deleteAtPosition(struct LinkedList *list, int position)  {
    if (position < 1 || list->head == NULL) {
```

```
        printf("Invalid position\n");
        return;
    }

    struct Node* current = list->head;
    struct Node* previous = NULL;

    if (position == 1) {
        deleteAtBeginning(list);
        return;
    }

    for (int i = 1; i < position && current->next != list->head; ++i) {
        previous = current;
        current = current->next;
    }

    if (current == list->head) {
        printf("Invalid position\n");
        return;
    }

    previous->next = current->next;
    free(current);
}
```

```
// Function to delete the node at the end of the linked list
void deleteAtEnd(struct LinkedList *list)  {
    if (list->head == NULL) {
        printf("List is empty\n");
        return;
    }

    if (list->head->next == list->head) {
        free(list->head);
        list->head = NULL;
        return;
    }

    struct Node* current = list->head;
    struct Node* previous = NULL;

    while (current->next != list->head) {
        previous = current;
        current = current->next;
    }

    previous->next = current->next;
    free(current);
```

```
}



// Function to display the linked list
void display(struct LinkedList* list) {
    struct Node* current = list->head;
    printf("Linked List: ");
    if (current != NULL) {
        do {
            printf("%d -> ", current->data);
            current = current->next;
        } while (current != list->head);
    }
    printf("HEAD\n");
}
```

Output:

```
######################
TEST CASE 1
######################

Menu:
1. Add at beginning
2. Add at any position
3. Add at end
4. Delete at beginning
5. Delete at any position
6. Delete at end
0. Quit
Enter your choice: 1
Enter data: 10Linked List: 10 -> HEAD

Menu:
1. Add at beginning
2. Add at any position
3. Add at end
4. Delete at beginning
5. Delete at any position
6. Delete at end
0. Quit
```

Enter your choice: 3

Enter data: 20Linked List: 10 -> 20 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 30Linked List: 10 -> 20 -> 30 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 2

Enter data: 25

Enter position: 2Linked List: 10 -> 25 -> 20 -> 30 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 40Linked List: 10 -> 25 -> 20 -> 30 -> 40 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 25 -> 20 -> 30 -> 40 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 5

Enter position: 2Linked List: 25 -> 30 -> 40 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 6Linked List: 25 -> 30 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: 30 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 4Linked List: HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 1

Enter data: 100Linked List: 100 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 3

Enter data: 200Linked List: 100 -> 200 -> HEAD

Menu:

1. Add at beginning

2. Add at any position

3. Add at end

4. Delete at beginning

5. Delete at any position

6. Delete at end

0. Quit

Enter your choice: 0

Exiting program.

3. Create a doubly connected linked list (DLL) in C. Write a menu driven program to perform

- **a. addition at the beginning**
- **b. addition at any given position**
- **c. addition at the end of the list**
- **d. deletion at the beginning**
- **e. deletion at any given position**
- **f. deletion at the end of the list**

Also create a display function that displays the entire linked list and call this function after every operation listed above.

The program should keep on displaying the menu to the user and wait for user's choice.

The user should be able to quit the program with choice 0.

Code:

```
#include <stdio.h>

#include <stdlib.h>


// Structure to represent a node in the doubly linked list

typedef struct Node {

  int data;

  struct Node* prev;

  struct Node* next;

} Node;


// Function to create a new node with the given data

Node* createNode(int data) {

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

  if (newNode == NULL) {

    printf("Memory allocation failed.\n");

    exit(EXIT_FAILURE);

  }
```

```
  newNode->data = data;
  newNode->prev = NULL;
  newNode->next = NULL;
  return newNode;
}

// Function to add a node at the beginning of the list
void addAtBeginning(Node** head, int data) {
  Node* newNode = createNode(data);
  if (*head == NULL) {
    *head = newNode;
  } else {
    newNode->next = *head;
    (*head)->prev = newNode;
    *head = newNode;
  }
}

// Function to add a node at a given position in the list
void addAtPosition(Node** head, int data, int position) {
  Node* newNode = createNode(data);
  if (position == 1) {
    newNode->next = *head;
    (*head)->prev = newNode;
    *head = newNode;
```

```
  } else {
    Node* current = *head;
    int i;
    for (i = 1; i < position - 1 && current != NULL; ++i) {
      current = current->next;
    }

    if (current == NULL) {
      printf("Invalid position. Cannot add at the specified position.\n");
      return;
    }

    newNode->next = current->next;
    newNode->prev = current;
    if (current->next != NULL) {
      current->next->prev = newNode;
    }
    current->next = newNode;
  }
}

// Function to add a node at the end of the list
void addAtEnd(Node** head, int data) {
  Node* newNode = createNode(data);
  if (*head == NULL) {
```

```
    *head = newNode;
  } else {
    Node* current = *head;
    while (current->next != NULL) {
      current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
  }
}

// Function to delete the node at the beginning of the list
void deleteAtBeginning(Node** head) {
  if (*head == NULL) {
    printf("List is empty. Cannot delete.\n");
    return;
  }

  Node* temp = *head;
  *head = temp->next;
  if (*head != NULL) {
    (*head)->prev = NULL;
  }
  free(temp);
}
```

```
// Function to delete a node at a given position in the list
void deleteAtPosition(Node** head, int position) {
  if (*head == NULL) {
    printf("List is empty. Cannot delete.\n");
    return;
  }

  Node* current = *head;
  int i;
  for (i = 1; i < position && current != NULL; ++i) {
    current = current->next;
  }

  if (current == NULL) {
    printf("Invalid position. Cannot delete at the specified position.\n");
    return;
  }

  if (current->prev != NULL) {
    current->prev->next = current->next;
  } else {
    *head = current->next;
  }
```

```
  if (current->next != NULL) {
    current->next->prev = current->prev;
  }

  free(current);
}

// Function to delete the node at the end of the list
void deleteAtEnd(Node** head) {
  if (*head == NULL) {
    printf("List is empty. Cannot delete.\n");
    return;
  }

  Node* current = *head;
  while (current->next != NULL) {
    current = current->next;
  }

  if (current->prev != NULL) {
    current->prev->next = NULL;
  } else {
    *head = NULL;
  }
```

```
  free(current);
}

// Function to display the entire linked list
void displayList(Node* head) {
  Node* current = head;
  printf("List: ");
  while (current != NULL) {
    printf("%d ", current->data);
    current = current->next;
  }
  printf("\n");
}

// Function to count the number of nodes in the list
int count_nodes(Node* head) {
  int count = 0;
  Node* current = head;
  while (current != NULL) {
    ++count;
    current = current->next;
  }
  return count;
}
```

```
int main() {
  Node *head = NULL; // Initialize the head of the list to NULL
  int choice = 1;
  int data, position;
  int i, testCases;
  /*
  IMPORTANT

  DO NOT CHANGE THE BELOW CODE.
  WRITE YOUR OWN CODE FOR FUNCTIONS (See comments after the main program).
  WRITE YOUR FUNCTIONS IN SUCH A WAY THAT
  THEY WILL WORK WITH THE BELOW CODE IN MAIN FUNCTION.

  TRY TO GET AN IDEA ABOUT THE FUNCTION PARAMETERS FROM THE FUNCTION CALLS IN
  THE MAIN FUNCTION.
  */
  scanf("%d", &testCases);
  for (i = 0; i < testCases; i++) {
    printf("\n##################\nTEST CASE %d\n##################\n", i + 1);
    do {
      printf("Menu:\n");
      printf("1. Add at beginning\n");
      printf("2. Add at position\n");
      printf("3. Add at end\n");
      printf("4. Delete at beginning\n");
```

```
printf("5. Delete at position\n");
printf("6. Delete at end\n");
printf("0. Quit\n");
scanf("%d", &choice);
printf("%d", choice);
switch (choice) {
case 1:
  printf("\nEnter the data to add: ");
  scanf("%d", &data);
  printf("%d", data);
  addAtBeginning(&head, data);
  displayList(head);
  break;
case 2:
  printf("\nEnter the data to add: ");
  scanf("%d", &data);
  printf("%d", data);
  printf("\nEnter the position to add: ");
  scanf("%d", &position);
  printf("%d", position);
  if (position < 1 || position > count_nodes(head)) {
    printf("\nInvalid position. Please try again.\n");
    break;
  }
  addAtPosition(&head, data, position);
```

```
  displayList(head);
  break;
 case 3:
  printf("\nEnter the data to add: ");
  scanf("%d", &data);
  printf("%d", data);
  addAtEnd(&head, data);
  displayList(head);
  break;
 case 4:
  if (head == NULL) {
    printf("\nList is empty. Cannot delete.\n");
    break;
  }
  deleteAtBeginning(&head);
  displayList(head);
  break;
 case 5:
  if (head == NULL) {
    printf("\nList is empty. Cannot delete.\n");
    break;
  }
  printf("\nEnter the position to delete: ");
  scanf("%d", &position);
  printf("%d", position);
```

```
      if (position < 1 || position > count_nodes(head)) {
        printf("n\Invalid position. Please try again.\n");
        break;
      }
      deleteAtPosition(&head, position);
      displayList(head);
      break;
    case 6:
      if (head == NULL) {
        printf("\nList is empty. Cannot delete.\n");
        break;
      }
      deleteAtEnd(&head);
      displayList(head);
      break;
    case 0:
      break;
    default:
      printf("\nInvalid choice\n");
      break;
    }
  } while (choice != 0);
}
  // Free the memory occupied by the list
  Node* current = head;
```

```
    Node* next;
    while (current != NULL) {
      next = current->next;
      free(current);
      current = next;
    }

    return 0;
}
```

Output:

```
#################
TEST CASE 1
#################
Menu:
1. Add at beginning
2. Add at position
3. Add at end
4. Delete at beginning
5. Delete at position
6. Delete at end
0. Quit
1
Enter the data to add: 10List: 10
```

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 20List: 10 20

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 30List: 10 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

2

Enter the data to add: 25

Enter the position to add: 2List: 10 25 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 40List: 10 25 20 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 25 20 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

5

Enter the position to delete: 2List: 25 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

6List: 25 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List:

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

1

Enter the data to add: 100List: 100

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 200List: 100 200

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

0

#################

TEST CASE 2

#################

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

1

Enter the data to add: 10List: 10 100 200

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 20List: 10 100 200 20

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 30List: 10 100 200 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

2

Enter the data to add: 25

Enter the position to add: 20

Invalid position. Please try again.

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 40List: 10 100 200 20 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 100 200 20 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

5

Enter the position to delete: 2List: 100 20 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

6List: 100 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

1

Enter the data to add: 100List: 100 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 200List: 100 30 200

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

0

#################

TEST CASE 3

#################

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

1

Enter the data to add: 10List: 10 100 30 200

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 20List: 10 100 30 200 20

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 30List: 10 100 30 200 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

2

Enter the data to add: 25

Enter the position to add: 2List: 10 25 100 30 200 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 40List: 10 25 100 30 200 20 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 25 100 30 200 20 30 40

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

5

Enter the position to delete: 50nInvalid position. Please try again.

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

6List: 25 100 30 200 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 100 30 200 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

4List: 30 200 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

1

Enter the data to add: 100List: 100 30 200 20 30

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit

3

Enter the data to add: 200List: 100 30 200 20 30 200

Menu:

1. Add at beginning

2. Add at position

3. Add at end

4. Delete at beginning

5. Delete at position

6. Delete at end

0. Quit