```
//Doubly Linked List
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;
   struct node *prev;
   struct node *next;
} *head = NULL;

// Create the doubly linked list
void create() {
   int num;
   struct node *newnode, *last = NULL;
   while (1) {
      printf("Enter a number (-1 to stop): ");
      scanf("%d", &num);
      if (num == -1)
         break;

      newnode = (struct node *)malloc(sizeof(struct node));
      if (newnode == NULL) {
         printf("Memory allocation failed.\n");
         return;
      }

      newnode->data = num;
      newnode->prev = NULL;
      newnode->next = NULL;

      if (head == NULL) {
         head = newnode;
      } else {
         last->next = newnode;
         newnode->prev = last;
      }
      last = newnode;
   }
}

// Insert at beginning
void insertAtBeginning(int data) {
   struct node *newnode = (struct node *)malloc(sizeof(struct node));
   if (newnode == NULL) {
      printf("Memory allocation failed!\n");
      return;
   }
   newnode->data = data;
   newnode->prev = NULL;
   newnode->next = head;
```

```
 if (head != NULL)
    head->prev = newnode;
    head = newnode;
    printf("Inserted %d at the beginning.\n", data);
}

// Insert at end
void insertAtEnd(int data) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newnode->data = data;
    newnode->next = NULL;

    if (head == NULL) {
        newnode->prev = NULL;
        head = newnode;
    } else {
        struct node *temp = head;
        while (temp->next != NULL)
            temp = temp->next;

        temp->next = newnode;
        newnode->prev = temp;
    }
    printf("Inserted %d at the end.\n", data);
}

// Insert at specific position
void insertAtPosition(int data, int position) {
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newnode->data = data;
    newnode->next = NULL;
    newnode->prev = NULL;

    if (position == 0) {
        newnode->next = head;
        if (head != NULL)
            head->prev = newnode;
        head = newnode;
        printf("Inserted %d at position 0.\n", data);
        return;
    }
    struct node *temp = head;
    for (int i = 0; temp != NULL && i < position - 1; i++)
        temp = temp->next;
```

```
    if (temp == NULL) {
        printf("Position out of bounds.\n");
        free(newnode);
        return;
    }

    newnode->next = temp->next;
    newnode->prev = temp;

    if (temp->next != NULL)
        temp->next->prev = newnode;

    temp->next = newnode;
    printf("Inserted %d at position %d.\n", data, position);
}

// Delete from beginning
void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct node *temp = head;
    head = head->next;

    if (head != NULL)
        head->prev = NULL;

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

// Delete from end
void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct node *temp = head;
    if (temp->next == NULL) {
        printf("Deleted %d from end.\n", temp->data);
        free(temp);
        head = NULL;
        return;
    }

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

    printf("Deleted %d from end.\n", temp->data);
    temp->prev->next = NULL;
```

```
    free(temp);
}

// Delete from specific position
void deleteAtPosition(int position) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    if (position == 0) {
        struct node *temp = head;
        head = head->next;
        if (head != NULL)
            head->prev = NULL;
        printf("Deleted %d from position 0.\n", temp->data);
        free(temp);
        return;
    }

    struct node *temp = head;
    for (int i = 0; temp != NULL && i < position; i++)
        temp = temp->next;

    if (temp == NULL) {
        printf("Position out of bounds.\n");
        return;
    }

    if (temp->prev != NULL)
        temp->prev->next = temp->next;

    if (temp->next != NULL)
        temp->next->prev = temp->prev;

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

// Display the list
void display() {
    struct node *temp = head;
    if (temp == NULL) {
        printf("List is empty.\n");
        return;
    }

    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
```

```
}

int main() {
    int mainChoice, subChoice, data, pos;

    while (1) {
        printf("\n--- Main Menu ---\n");
        printf("1. Create List\n");
        printf("2. Insert\n");
        printf("3. Delete\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &mainChoice);

        switch (mainChoice) {
            case 1:
                create();
                break;
            case 2:
                printf("\n-- Insert Menu --\n");
                printf("1. Insert at Beginning\n");
                printf("2. Insert at End\n");
                printf("3. Insert at Position\n");
                printf("Enter your choice: ");
                scanf("%d", &subChoice);
                printf("Enter data to insert: ");
                scanf("%d", &data);
                switch (subChoice) {
                    case 1: insertAtBeginning(data); break;
                    case 2: insertAtEnd(data); break;
                    case 3:
                        printf("Enter position: ");
                        scanf("%d", &pos);
                        insertAtPosition(data, pos);
                        break;
                    default: printf("Invalid insert choice.\n");
                }
                break;
            case 3:
                printf("\n-- Delete Menu --\n");
                printf("1. Delete from Beginning\n");
                printf("2. Delete from End\n");
                printf("3. Delete from Position\n");
                printf("Enter your choice: ");
                scanf("%d", &subChoice);
                switch (subChoice) {
                    case 1: deleteFromBeginning(); break;
                    case 2: deleteFromEnd(); break;
                    case 3:
                        printf("Enter position to delete: ");
                        scanf("%d", &pos);
                        deleteAtPosition(pos);
```

```
                break;
            default: printf("Invalid delete choice.\n");
        }
        break;
    case 4:
        display();
        break;
    case 5:
        printf("Exiting program.\n");
        exit(0);
    default:
        printf("Invalid choice. Please try again.\n");
  }
 }
 return 0;
}
```

Sample Output:

```
--- Main Menu ---
1. Create List
2. Insert
3. Delete
4. Display
5. Exit
Enter your choice: 1
Enter a number (-1 to stop): 4
Enter a number (-1 to stop): 6
Enter a number (-1 to stop): 7
Enter a number (-1 to stop): -1

--- Main Menu ---
1. Create List
2. Insert
3. Delete
4. Display
5. Exit
Enter your choice: 4
Doubly Linked List: 4 <-> 6 <-> 7 <-> NULL

--- Main Menu ---
1. Create List
```