```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* prev;
   struct Node* next;
};

struct Node* createNode(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL) {
      printf("Memory allocation failed\n");
      exit(EXIT_FAILURE);
   }
   newNode->data = data;
   newNode->prev = newNode->next = NULL;
   return newNode;
}

void insertNodeAtBeginning(struct Node** head, int data) {
   struct Node* newNode = createNode(data);

   if (*head == NULL) {
      *head = newNode;
   } else {
      newNode->next = *head;
      (*head)->prev = newNode;
      *head = newNode;
   }
}

void insertNodeAtEnd(struct Node** head, int data) {
   struct Node* newNode = createNode(data);

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

```
void deleteNode(struct Node** head, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = *head;

    if (current->data == data) {
        *head = current->next;
        if (*head != NULL) {
            (*head)->prev = NULL;
        }
        free(current);
    } else {
        while (current != NULL && current->data != data) {
            current = current->next;
        }

        if (current == NULL) {
            printf("Node with data %d not found\n", data);
            return;
        }

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

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

        free(current);
    }
}

void traverseForward(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = head;

    while (current != NULL) {
        printf("%d <-> ", current->data);
        current = current->next;
```

```
  }

  printf("NULL\n");
}

void traverseBackward(struct Node* head) {
  if (head == NULL) {
    printf("List is empty\n");
    return;
  }

  struct Node* current = head;

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

  while (current != NULL) {
    printf("%d <-> ", current->data);
    current = current->prev;
  }

  printf("NULL\n");
}

void freeList(struct Node** head) {
  if (*head == NULL) {
    return;
  }

  struct Node *current = *head, *temp;

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

  *head = NULL;
}

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

  do {
    printf("\n1. Insert Node at Beginning\n");
```

```
        printf("2. Insert Node at End\n");
        printf("3. Delete Node\n");
        printf("4. Traverse Forward\n");
        printf("5. Traverse Backward\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data for the new node: ");
                scanf("%d", &data);
                insertNodeAtBeginning(&head, data);
                break;
            case 2:
                printf("Enter data for the new node: ");
                scanf("%d", &data);
                insertNodeAtEnd(&head, data);
                break;
            case 3:
                printf("Enter data of the node to delete: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 4:
                printf("Doubly Linked List (Forward):\n");
                traverseForward(head);
                break;
            case 5:
                printf("Doubly Linked List (Backward):\n");
                traverseBackward(head);
                break;
            case 0:
                printf("Exiting the program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 0);

    // Free the memory allocated for the doubly linked list
    freeList(&head);

    return 0;
}
```