Doubly Linked List Operations

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

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

struct Node* head = NULL;

// Function to insert a node at the beginning
void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = head;

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

    head = newNode;
}

// Function to insert a node at the end
void insertAtEnd(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

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

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

```
        current = current->next;
    }

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

// Function to insert a node at a specific position
void insertAtPosition(int value, int position) {
    if (position <= 0) {
        printf("Invalid position\n");
        return;
    }

    if (position == 1) {
        insertAtBeginning(value);
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    struct Node* current = head;
    for (int i = 1; i < position - 1; ++i) {
        if (current == NULL) {
            printf("Position out of bounds\n");
            free(newNode);
            return;
        }
        current = current->next;
    }

    if (current == NULL) {
        printf("Position out of bounds\n");
        free(newNode);
        return;
    }

    newNode->prev = current;
    newNode->next = current->next;

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

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

// Function to delete a node from the beginning
void deleteAtBeginning() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

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

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

    free(temp);
}

// Function to delete a node from the end
void deleteAtEnd() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct 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 delete a node from a specific position
```

```
void deleteAtPosition(int position) {
    if (position <= 0 || head == NULL) {
        printf("Invalid position or empty list\n");
        return;
    }

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

    struct Node* current = head;
    for (int i = 1; i < position; ++i) {
        if (current == NULL) {
            printf("Position out of bounds\n");
            return;
        }
        current = current->next;
    }

    if (current == NULL) {
        printf("Position out of
        bounds\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 search for a value in the list
int search(int value) {
    struct Node* current = head;
    int position = 1;
```

```
    while (current != NULL) {
        if (current->data == value) {
            return position;
        }
        current = current->next;
        position++;
    }

    return -1;
}

// Function to display the list
void display() {
    struct Node* current = head;

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

    printf("\n");
}

int main() {
    insertAtBeginning(5);
    insertAtEnd(10);
    insertAtEnd(15);
    insertAtBeginning(2);

    printf("List after inserting at beginning and end:\n");
    display();

    insertAtPosition(8, 3);
    printf("List after inserting at position 3:\n");
    display();

    deleteAtBeginning();
    printf("List after deleting at beginning:\n");
    display();

    deleteAtEnd();
    printf("List after deleting at end:\n");
```

```
    display();

    deleteAtPosition(2);
    printf("List after deleting at position 2:\n");
    display();
    int searchValue = 10;
    int position = search(searchValue);
    if (position != -1) {
        printf("%d found at position %d\n", searchValue, position);
    } else {
        printf("%d not found in the list\n",
    searchValue); }

    return 0;
}
```