4. Complete Singly Linked List Example

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

// Node structure
struct Node {
    int data;
    struct Node *next;
};

// Function prototypes
void insertAtBeginning(struct Node **head, int value);
void insertAtEnd(struct Node **head, int value);
void insertAtPosition(struct Node **head, int value, int position);
void deleteFromBeginning(struct Node **head);
void deleteFromEnd(struct Node **head);
void deleteAtPosition(struct Node **head, int position);
void deleteByValue(struct Node **head, int value);
void displayList(struct Node *head);
int search(struct Node *head, int value);
int getLength(struct Node *head);
void freeList(struct Node **head);

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

    printf("=== Linked List Operations Demo ===\n\n");

    // Insert operations
    printf("Inserting elements...\n");
    insertAtBeginning(&head, 10);
    insertAtBeginning(&head, 5);
    insertAtEnd(&head, 20);
    insertAtEnd(&head, 25);
    insertAtPosition(&head, 15, 2);
```

```
    printf("After insertions: ");
    displayList(head);
    printf("Length: %d\n\n", getLength(head));

    // Search operation
    int searchValue = 15;
    int pos = search(head, searchValue);
    if (pos != -1) {
        printf("Value %d found at position %d\n\n", searchValue, pos);
    } else {
        printf("Value %d not found\n\n", searchValue);
    }

    // Delete operations
    printf("Deleting from beginning...\n");
    deleteFromBeginning(&head);
    displayList(head);

    printf("Deleting from end...\n");
    deleteFromEnd(&head);
    displayList(head);

    printf("Deleting at position 1...\n");
    deleteAtPosition(&head, 1);
    displayList(head);

    printf("Deleting value 10...\n");
    deleteByValue(&head, 10);
    displayList(head);

    printf("\nFinal length: %d\n", getLength(head));

    // Clean up
    freeList(&head);
    printf("Memory freed.\n");

    return 0;
}

// Function implementations
```

```
void insertAtBeginning(struct Node **head, int value) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void insertAtEnd(struct Node **head, int value) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;

    if (*head == NULL) {
        *head = newNode;
        return;
    }

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

void insertAtPosition(struct Node **head, int value, int position) {
    struct Node *newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = value;

    if (position == 0) {
```

```
        newNode->next = *head;
        *head = newNode;
        return;
    }

    struct Node *temp = *head;
    for (int i = 0; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Invalid position!\n");
        free(newNode);
        return;
    }

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

void deleteFromBeginning(struct Node **head) {
    if (*head == NULL) {
        printf("List is empty!\n");
        return;
    }
    struct Node *temp = *head;
    *head = (*head)->next;
    free(temp);
}

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

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

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

void deleteAtPosition(struct Node **head, int position) {
    if (*head == NULL) {
        printf("List is empty!\n");
        return;
    }

    if (position == 0) {
        struct Node *temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
    }

    struct Node *temp = *head;
    for (int i = 0; i < position - 1 && temp != NULL; i++) {
        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        printf("Invalid position!\n");
        return;
    }

    struct Node *nodeToDelete = temp->next;
    temp->next = nodeToDelete->next;
    free(nodeToDelete);
}

void deleteByValue(struct Node **head, int value) {
    if (*head == NULL) {
        printf("List is empty!\n");
        return;
```

```
    }

    if ((*head)->data == value) {
        struct Node *temp = *head;
        *head = (*head)->next;
        free(temp);
        return;
    }

    struct Node *temp = *head;
    while (temp->next != NULL && temp->next->data != value) {
        temp = temp->next;
    }

    if (temp->next == NULL) {
        printf("Value %d not found!\n", value);
        return;
    }

    struct Node *nodeToDelete = temp->next;
    temp->next = nodeToDelete->next;
    free(nodeToDelete);
}

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

    struct Node *temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf(" → ");
        }
        temp = temp->next;
    }
    printf(" → NULL\n");
}
```

```
int search(struct Node *head, int value) {
    struct Node *temp = head;
    int position = 0;

    while (temp != NULL) {
        if (temp->data == value) {
            return position;
        }
        temp = temp->next;
        position++;
    }
    return -1;
}

int getLength(struct Node *head) {
    int count = 0;
    struct Node *temp = head;

    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    return count;
}

void freeList(struct Node **head) {
    struct Node *temp;
    while (*head != NULL) {
        temp = *head;
        *head = (*head)->next;
        free(temp);
    }
}
```

Revision #1
Created 2025-10-27 05:02:35 UTC by DS
Updated 2025-10-27 05:02:50 UTC by DS