Doubly Linked Lists

Kishoreadhith V - 22z232

Functions

1. Structure and function to create node in a doubly linked list

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

struct Node * getNode(int val){
    struct Node * newNode = (struct Node *) malloc(sizeof(struct Node
*));
    newNode->data = val;
    newNode->next = NULL;
    newNode->prev = NULL;
}
```

Usage:

```
struct Node * newNode = getNode(10);
```

2. Function to insert a new node at the beginning of a doubly linked list

PROF

```
void insertFirst(struct Node ** head, int val){
    if (*head == NULL)
    {
    struct Node * newNode = getNode(val);
    *head = newNode;
    }
    struct Node * newNode = getNode(val);
    newNode->next = *head;
    newNode->next->prev = newNode;
    *head = newNode;
}
```

Usage:

```
insertFirst(&head, 10);
```

3. Function to insert a new node at the end of a doubly linked list

```
void InsertLast(struct Node ** head, int val){
    if (*head == NULL)
    {
        struct Node * newNode = getNode(val);
        *head = newNode;
        return;
    }
    struct Node * newNode = getNode(val);
    struct Node * current = *head;
    while (current->next != NULL)
    {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
}
```

Usage:

```
insertLast(&head, 10);
```

4. Function to display a linked list

```
void display(struct Node * head){
    struct Node *current = head;
    printf("---\n");
    printf("Head: %p\n", head);
    printf("---\n");
    if (head == NULL)
    {
        printf("List is empty\n");
        printf("---\n");
        return;
    }

    while (current != NULL) {
        printf("Address: %p\n", current);
        printf("Data: %d\n", current->data);
        printf("Prev: %p\n", current->prev);
        printf("Next: %p\n", current->next);
        printf("---\n");
        current = current->next;
```

PROF

```
    }
}
```

Usage:

```
display(&head);
```

5. Function to search for a node based on its value

```
struct Node * search(struct Node ** head, int target){
    if (*head == NULL)
    {
        printf("List is empty\n");
        printf("---\n");
        return NULL;
    }
    struct Node *current = *head;
    while (current != NULL)
    {
        if (current->data == target)
        {
            return current;
        }
        current = current->next;
    }
    return NULL;
}
```

Usage:

PROF

```
struct Node *result = search(&head, 10);
if (result != NULL)
{
    printf("Found at %p\n", result);
}
else
{
    printf("Not found\n");
}
```

6. Function to insert a new node after a specified node with a given value in a doubly linked list

```
void insertAfter(struct Node ** head, int target, int val){
    struct Node * prev = search(head, target);
    struct Node * newNode = getNode(val);
    newNode->prev = prev;
    newNode->next = prev->next;
    prev->next = newNode;
    newNode->next->prev = newNode;
}
```

Usage:

```
insertAfter(&head, 10, 20);
```

7. Function to insert a new node before a specified node with a given value in a doubly linked list

```
void insertBefore(struct Node ** head,int target, int data){
    struct Node * next = search(head, target);
    struct Node *newNode = getNode(data);
    newNode->next = next;
    newNode->prev = next->prev;
    newNode->prev->next = newNode;
    next->prev = newNode;
}
```

Usage:

```
insertBefore(&head, 10, 20);
```

8. Function to delete a node with a given value

```
void deleteNode(struct Node * node){
    if (node == NULL)
    {
        printf("Node not found\n");
        return;
    }
    if (node->next == NULL)
    {
        node->prev->next = NULL;
        free(node);
        return;
    }
```

```
        node->prev->next = node->next;
        node->next->prev = node->prev;
        free(node);
    }

    void deleteNodeOf(struct Node ** head, int val){
        struct Node * del = search(head, val);
        deleteNode(del);
    }
```

Usage:

```
deleteNode(&head, 10);
```

9. Function to delete the first node

```
void deleteFirst(struct Node ** head){
    if(*head == NULL){
        return;
    }
    struct Node * del = *head;
    *head = del->next;
    (*head)->prev = NULL;
    free(del);
}
```

Usage:

```
deleteFirst(&head);
```

PROF

10. Function to delete the last node

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

Usage:

```
deleteLast(&head);
```

11. Function to delete the node after the node with a given value

```
void deleteAfter(struct Node ** head, int target){
    struct Node * prev = search(head, target);
    deleteNode(prev->next);
}
```

Usage:

```
deleteAfter(&head, 10);
```

12. Function to delete the node before the node with a given value

```
void deleteBefore(struct Node ** head, int target){
    struct Node * next = search(head, target);
    deleteNode(next->prev);
}
```

Usage:

```
deleteBefore(&head, 10);
```

PROF