Kartheeka Repalle

192372289

CSE-AI

25/07/2024

LINKED LIST:

Linked List is a linear data structure, that store the collection of elements in the form of Nodes. Linked List forms a series of connected nodes, where each node stores the data and the address of the next node.

Types of linked list:

There are mainly three types of linked lists:

1. Singly-linked list
2. Doubly linked list
3. Circular linked list

1. **Single linked list:**

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

// Define the structure for a linked list node
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node at the head of the list
void insertAtHead(Node** head, int data) {
    Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

// Function to print the linked list
```

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

// Function to search for a node in the list
Node* searchNode(Node* head, int data) {
    Node* temp = head;
    while (temp != NULL) {
        if (temp->data == data) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

// Function to delete a node from the list
void deleteNode(Node** head, Node* nodeToDelete) {
    if (*head == nodeToDelete) {
        *head = nodeToDelete->next;
    } else {
        Node* temp = *head;
        while (temp->next != nodeToDelete) {
            temp = temp->next;
        }
        temp->next = nodeToDelete->next;
    }
    free(nodeToDelete);
}

// Function to update the data of a node
void updateNode(Node* nodeToUpdate, int newData) {
    nodeToUpdate->data = newData;
}

int main() {
    Node* head = NULL;

    // Insert nodes into the list
    insertAtHead(&head, 1);
    insertAtHead(&head, 2);
    insertAtHead(&head, 3);
```

```
    // Print the linked list
    printf("Linked List: ");
    printList(head);

    // Search for a node
    Node* searchedNode = searchNode(head, 2);
    if (searchedNode != NULL) {
        printf("Node with data %d found\n", searchedNode->data);
    } else {
        printf("Node not found\n");
    }

    // Delete a node
    deleteNode(&head, searchedNode);

    // Print the linked list after deletion
    printf("Linked List after deletion: ");
    printList(head);

    // Update the data of a node
    Node* nodeToUpdate = searchNode(head, 1);
    updateNode(nodeToUpdate, 10);

    // Print the linked list after update
    printf("Linked List after update: ");
    printList(head);

    return 0;
}
```

Output:

```
single Linked List: 3 2 1
Node with data 2 found
single Linked List after deletion: 3 1
single Linked List after update: 3 10
```

2. Double linked list:

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

// Define the structure for a double linked list node
typedef struct Node {
    int data;
```

```
    struct Node* next;
    struct Node* prev;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

// Function to insert a new node at the head of the list
void insertAtHead(Node** head, int data) {
    Node* newNode = createNode(data);
    newNode->next = *head;
    if (*head != NULL) {
        (*head)->prev = newNode;
    }
    *head = newNode;
}

// Function to print the double linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Function to search for a node in the list
Node* searchNode(Node* head, int data) {
    Node* temp = head;
    while (temp != NULL) {
        if (temp->data == data) {
            return temp;
        }
        temp = temp->next;
    }
    return NULL;
}

// Function to delete a node from the list
void deleteNode(Node** head, Node* nodeToDelete) {
```

```
    if (*head == nodeToDelete) {
        *head = nodeToDelete->next;
    } else {
        nodeToDelete->prev->next = nodeToDelete->next;
    }
    if (nodeToDelete->next != NULL) {
        nodeToDelete->next->prev = nodeToDelete->prev;
    }
    free(nodeToDelete);
}

// Function to update the data of a node
void updateNode(Node* nodeToUpdate, int newData) {
    nodeToUpdate->data = newData;
}

int main() {
    Node* head = NULL;

    // Insert nodes into the list
    insertAtHead(&head, 1);
    insertAtHead(&head, 2);
    insertAtHead(&head, 3);

    // Print the double linked list
    printf("Double Linked List: ");
    printList(head);

    // Search for a node
    Node* searchedNode = searchNode(head, 2);
    if (searchedNode != NULL) {
        printf("Node with data %d found\n", searchedNode->data);
    } else {
        printf("Node not found\n");
    }

    // Delete a node
    deleteNode(&head, searchedNode);

    // Print the double linked list after deletion
    printf("Double Linked List after deletion: ");
    printList(head);

    // Update the data of a node
    Node* nodeToUpdate = searchNode(head, 1);
    updateNode(nodeToUpdate, 10);
```

```
    // Print the double linked list after update
    printf("Double Linked List after update: ");
    printList(head);

    return 0;
}
```

Output:

```
Double Linked List: 3 2 1
Node with data 2 found
Double Linked List after deletion: 3 1
Double Linked List after update: 3 10
```

3. Circular linked list:

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

// Define the structure for a circular linked list node
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a new node at the head of the list
void insertAtHead(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;
        newNode->next = newNode;
    } else {
        newNode->next = *head;
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
```

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

// Function to print the circular linked list
void printList(Node* head) {
    Node* temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("\n");
}

// Function to search for a node in the list
Node* searchNode(Node* head, int data) {
    Node* temp = head;
    do {
        if (temp->data == data) {
            return temp;
        }
        temp = temp->next;
    } while (temp != head);
    return NULL;
}

// Function to delete a node from the list
void deleteNode(Node** head, Node* nodeToDelete) {
    if (*head == nodeToDelete) {
        Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }
        temp->next = nodeToDelete->next;
        *head = nodeToDelete->next;
    } else {
        Node* temp = *head;
        while (temp->next != nodeToDelete) {
            temp = temp->next;
        }
        temp->next = nodeToDelete->next;
    }
    free(nodeToDelete);
}
```

```
// Function to update the data of a node
void updateNode(Node* nodeToUpdate, int newData) {
    nodeToUpdate->data = newData;
}

int main() {
    Node* head = NULL;

    // Insert nodes into the list
    insertAtHead(&head, 1);
    insertAtHead(&head, 2);
    insertAtHead(&head, 3);

    // Print the circular linked list
    printf("Circular Linked List: ");
    printList(head);

    // Search for a node
    Node* searchedNode = searchNode(head, 2);
    if (searchedNode != NULL) {
        printf("Node with data %d found\n", searchedNode->data);
    } else {
        printf("Node not found\n");
    }

    // Delete a node
    deleteNode(&head, searchedNode);

    // Print the circular linked list after deletion
    printf("Circular Linked List after deletion: ");
    printList(head);

    // Update the data of a node
    Node* nodeToUpdate = searchNode(head, 1);
    updateNode(nodeToUpdate, 10);

    // Print the circular linked list after update
    printf("Circular Linked List after update: ");
    printList(head);

    return 0;
}
```

Output:

```
Circular Linked List: 3 2 1
Node with data 2 found
```

Circular Linked List after deletion: 3 1
Circular Linked List after update: 3 10