

Lesson 1: Introduction to Linked List

A **linked list** is a linear data structure where elements (called nodes) are stored in memory, but unlike arrays, they are not stored in contiguous memory locations. Each node contains:

1. **Data** (stores the actual value)
2. **Pointer (next)** (stores the address of the next node)

Types of Linked Lists

1. **Singly Linked List** (each node points to the next node)
2. **Doubly Linked List** (each node points to both previous and next node)
3. **Circular Linked List** (last node points back to the first node)

We'll focus on **Singly Linked Lists** first.

Lesson 2: Creating a Node in a Singly Linked List

In C++, we define a node using a `struct` or a `class`.

Code: Defining a Node

cpp

```
#include <iostream>
using namespace std;

// Node structure
struct Node {
    int data;        // Data
    Node* next;     // Pointer to the next node
};

// Function to create a new node
Node* createNode(int value) {
    Node* newNode = new Node(); // Allocate memory for new node
    newNode->data = value;       // Assign data
    newNode->next = NULL;       // Set next pointer to NULL
    return newNode;
}

int main() {
```

```
Node* first = createNode(10);  
cout << "Node created with data: " << first->data << endl;  
return 0;  
}
```

Next Lesson: Insertion in Linked List

In the next lesson, we'll cover how to insert nodes at different positions (beginning, end, middle). Let me know if you want any modifications in the approach!

You said:

No continue

ChatGPT said:

Lesson 3: Insertion in a Singly Linked List

Insertion in a linked list can be done in three ways:

1. **At the beginning**
2. **At the end**
3. **At a specific position**

1. Insertion at the Beginning

We create a new node and make it the new head by pointing it to the previous first node.

Code: Insert at Beginning

cpp

```
#include <iostream>  
using namespace std;  
  
struct Node {  
    int data;  
    Node* next;  
};  
  
void insertAtBeginning(Node*& head, int value) {  
    Node* newNode = new Node();  
    newNode->data = value;  
    newNode->next = head;
```

```

    head = newNode;
}

void display(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

int main() {
    Node* head = NULL;

    insertAtBeginning(head, 10);
    insertAtBeginning(head, 20);
    insertAtBeginning(head, 30);

    display(head);
    return 0;
}

```

Output:

rust

```
30 -> 20 -> 10 -> NULL
```

2. Insertion at the End

We traverse to the last node and set its `next` pointer to the new node.

Code: Insert at End

cpp

```

void insertAtEnd(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = NULL;

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

```

```

    }

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

```

3. Insertion at a Specific Position

We traverse to the given position and adjust the pointers.

Code: Insert at a Specific Position

cpp

```

void insertAtPosition(Node*& head, int value, int pos) {
    Node* newNode = new Node();
    newNode->data = value;

    if (pos == 1) {
        newNode->next = head;
        head = newNode;
        return;
    }

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

    if (temp == NULL) {
        cout << "Position out of range!" << endl;
        return;
    }

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

```

Lesson 4: Deletion in a Singly Linked List

Deletion can also be done in three ways:

1. Delete from beginning

2. Delete from end

3. Delete a specific node

I'll continue with the deletion lesson next! Let me know if you have any doubts so far.

You said:

No doubts

ChatGPT said:

Lesson 4: Deletion in a Singly Linked List

Deletion operations involve updating pointers to remove a node from the linked list.

1. Deletion from the Beginning

We move the `head` to the next node and delete the previous head.

Code: Delete from Beginning

cpp

```
void deleteFromBeginning(Node*& head) {  
    if (head == NULL) {  
        cout << "List is empty!" << endl;  
        return;  
    }  
  
    Node* temp = head;  
    head = head->next;  
    delete temp;  
}
```

2. Deletion from the End

We traverse to the second-last node and set its `next` pointer to `NULL`, then delete the last node.

Code: Delete from End

cpp

```

void deleteFromEnd(Node*& head) {
    if (head == NULL) {
        cout << "List is empty!" << endl;
        return;
    }

    if (head->next == NULL) {
        delete head;
        head = NULL;
        return;
    }

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

    delete temp->next;
    temp->next = NULL;
}

```

3. Deletion from a Specific Position

We traverse to the node before the target node and adjust the `next` pointer.

Code: Delete from a Specific Position

cpp

```

void deleteAtPosition(Node*& head, int pos) {
    if (head == NULL) {
        cout << "List is empty!" << endl;
        return;
    }

    if (pos == 1) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* temp = head;
    for (int i = 1; temp != NULL && i < pos - 1; i++) {

```

```

        temp = temp->next;
    }

    if (temp == NULL || temp->next == NULL) {
        cout << "Position out of range!" << endl;
        return;
    }

    Node* deleteNode = temp->next;
    temp->next = deleteNode->next;
    delete deleteNode;
}

```

Lesson 5: Searching in a Singly Linked List

We traverse the list to find a value.

Code: Search in Linked List

cpp

```

bool search(Node* head, int key) {
    Node* temp = head;
    while (temp != NULL) {
        if (temp->data == key)
            return true;
        temp = temp->next;
    }
    return false;
}

```

Lesson 6: Traversing a Linked List

We print each node's data until we reach `NULL`.

Code: Traversal

cpp

```

void display(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
}

```

```
}  
cout << "NULL" << endl;  
}
```

Next Lesson: Reversing a Linked List

Next, we will cover how to **reverse** a linked list. Let me know if you're ready to continue!

ChatGPT can make mistakes. Check important info.