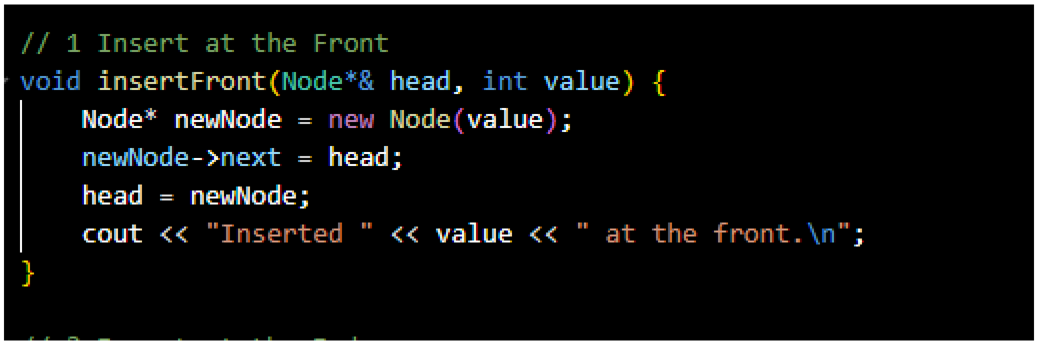
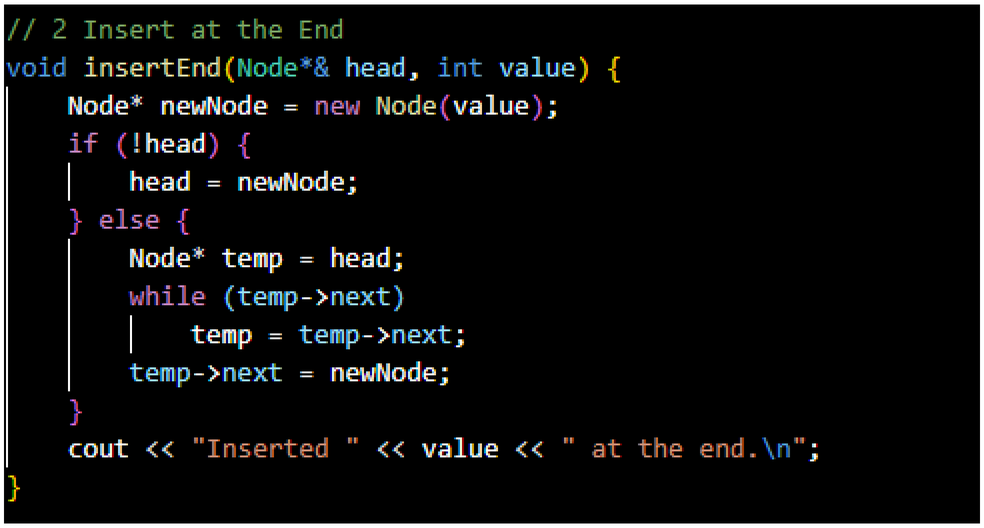
. Challenge 1 – insert at the front



```
// 1 Insert at the Front
void insertFront(Node*& head, int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
    cout << "Inserted " << value << " at the front.\n";
}
```

Explanation: make the new node point to old head

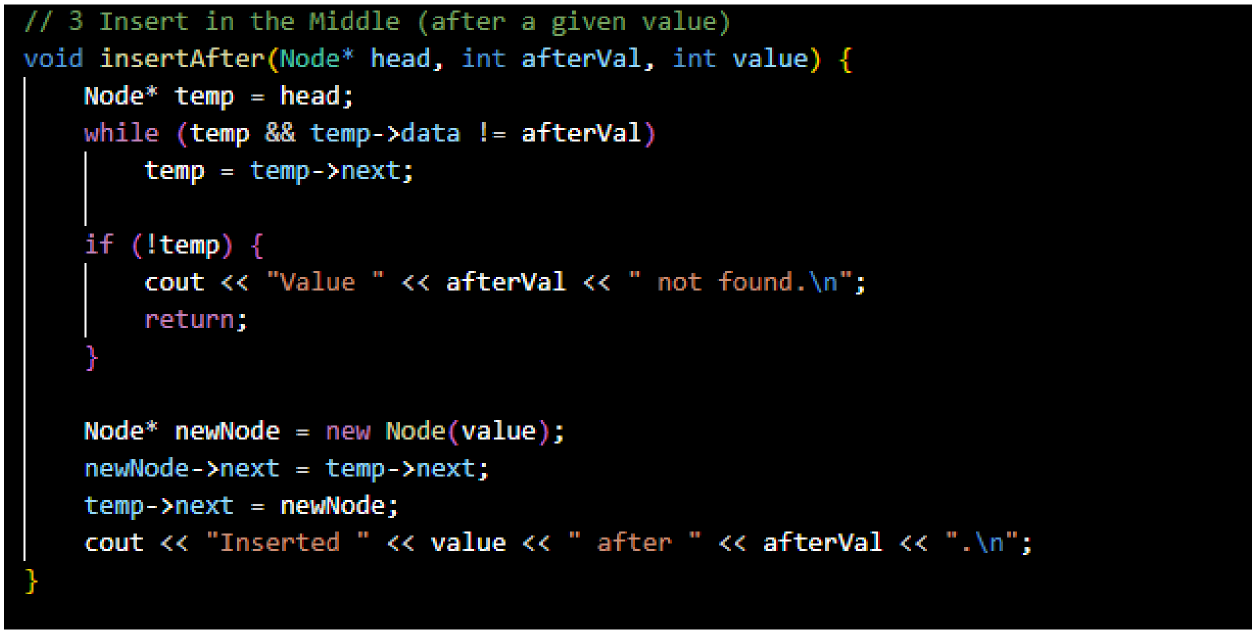
. Challenge 2 – insert at the end



```
// 2 Insert at the End
void insertEnd(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next)
            temp = temp->next;
        temp->next = newNode;
    }
    cout << "Inserted " << value << " at the end.\n";
}
```

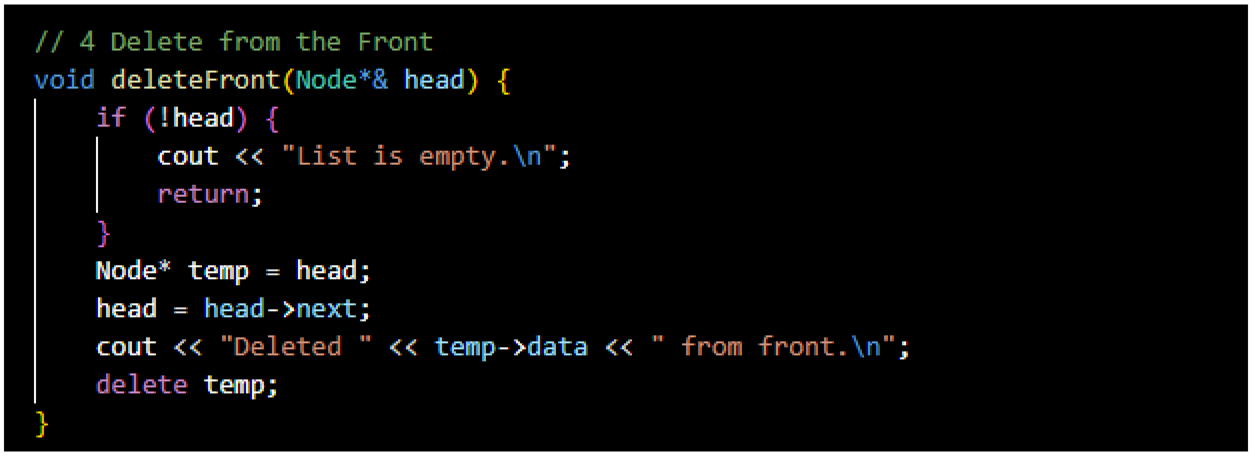
Explanation: traverses to the last node, link new on

. Challenge 2 – insert in the middle

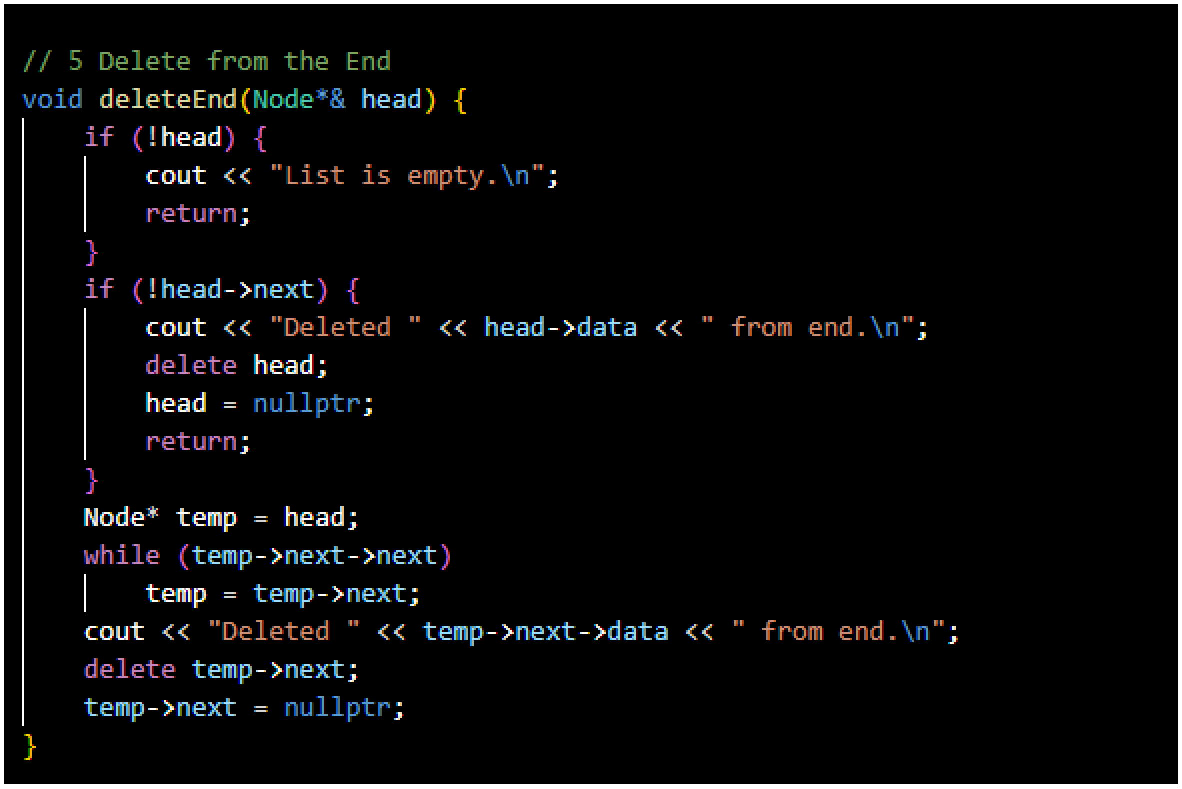

```cpp
// 3 Insert in the Middle (after a given value)
void insertAfter(Node* head, int afterVal, int value) {
    Node* temp = head;
    while (temp && temp->data != afterVal)
        temp = temp->next;

    if (!temp) {
        cout << "Value " << afterVal << " not found.\n";
        return;
    }

    Node* newNode = new Node(value);
    newNode->next = temp->next;
    temp->next = newNode;
    cout << "Inserted " << value << " after " << afterVal << ".\n";
}
```

Explanation: finds node with target value, insert next

Challenge 4- delete from the front


```cpp
// 4 Delete from the Front
void deleteFront(Node*& head) {
    if (!head) {
        cout << "List is empty.\n";
        return;
    }
    Node* temp = head;
    head = head->next;
    cout << "Deleted " << temp->data << " from front.\n";
    delete temp;
}
```
Explanation: move head to next node

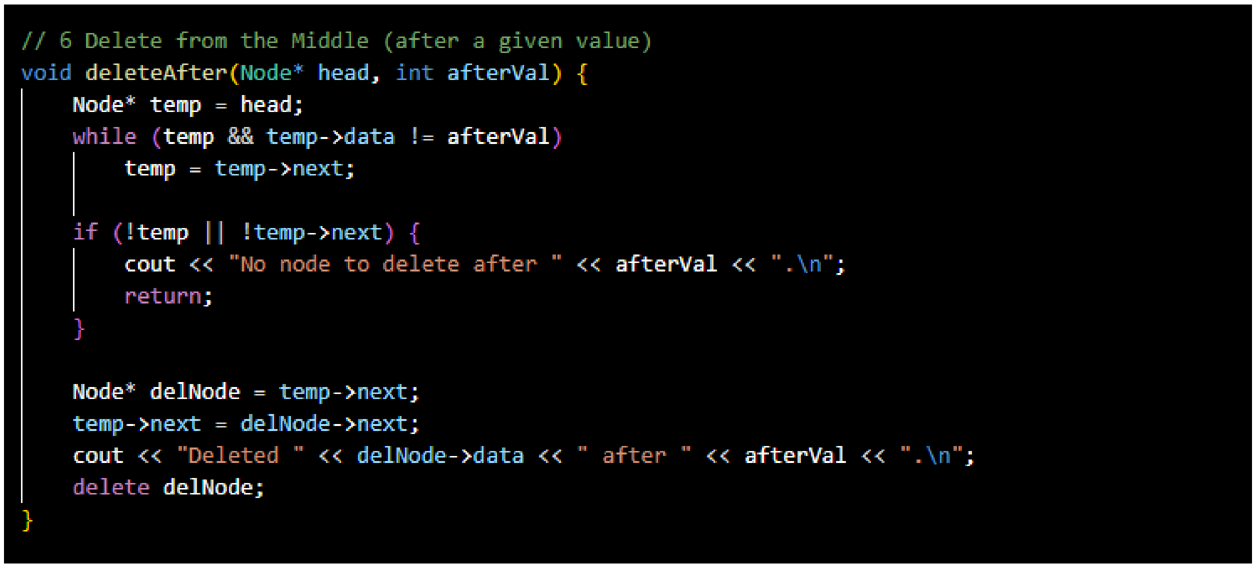
Challenge 5 – remove the last node



```cpp
// 5 Delete from the End
void deleteEnd(Node*& head) {
    if (!head) {
        cout << "List is empty.\n";
        return;
    }
    if (!head->next) {
        cout << "Deleted " << head->data << " from end.\n";
        delete head;
        head = nullptr;
        return;
    }
    Node* temp = head;
    while (temp->next->next)
        temp = temp->next;
    cout << "Deleted " << temp->next->data << " from end.\n";
    delete temp->next;
    temp->next = nullptr;
}
```

Explanation: find node before last and deletes last

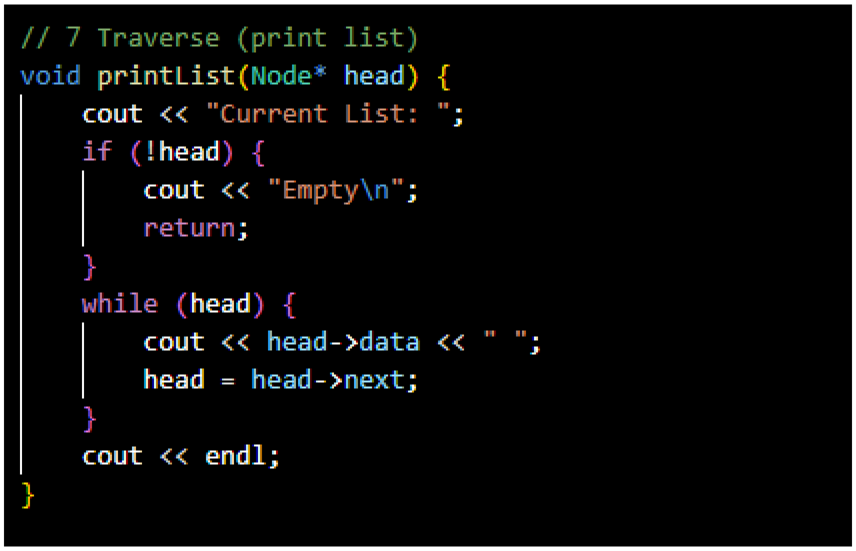
Challenge 6 – delete middle



```cpp
// 6 Delete from the Middle (after a given value)
void deleteAfter(Node* head, int afterVal) {
    Node* temp = head;
    while (temp && temp->data != afterVal)
        temp = temp->next;

    if (!temp || !temp->next) {
        cout << "No node to delete after " << afterVal << ".\n";
        return;
    }

    Node* delNode = temp->next;
    temp->next = delNode->next;
    cout << "Deleted " << delNode->data << " after " << afterVal << ".\n";
    delete delNode;
}
```

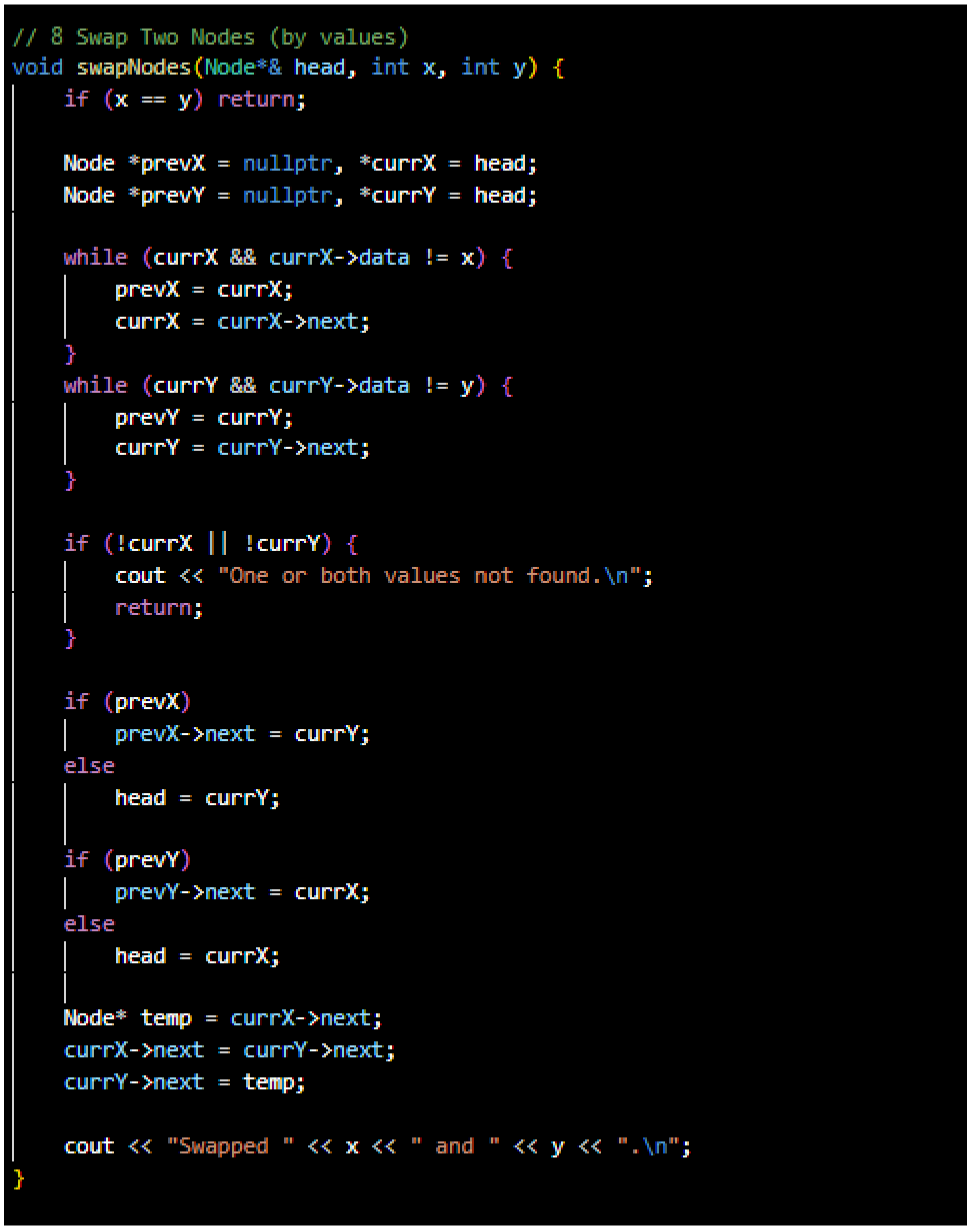
Explanation: skips the next pointer of the target point

Challenge 7 – traverse the list


```
// 7 Traverse (print list)
void printList(Node* head) {
    cout << "Current List: ";
    if (!head) {
        cout << "Empty\n";
        return;
    }
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
```

Explanation: loop through all nodes

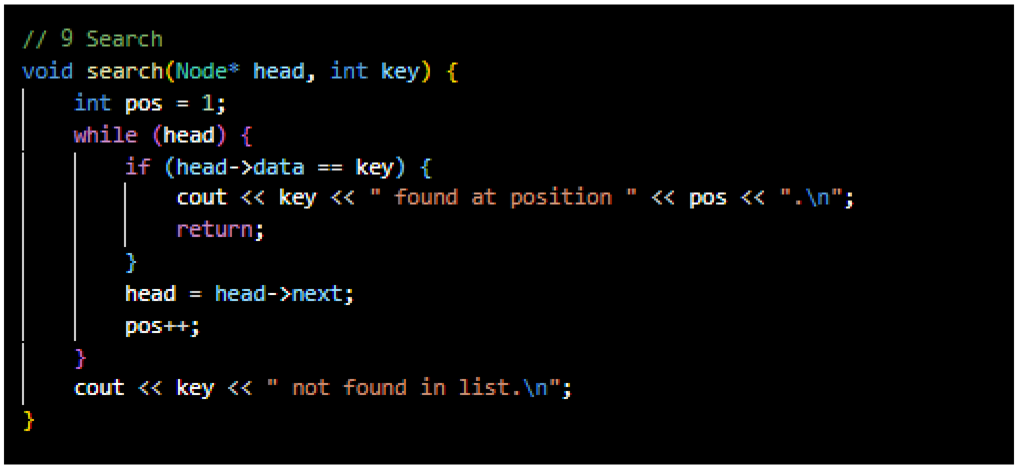
Challenge 8- swap two nodes



```cpp
// 8 Swap Two Nodes (by values)
void swapNodes(Node*& head, int x, int y) {
    if (x == y) return;

    Node *prevX = nullptr, *currX = head;
    Node *prevY = nullptr, *currY = head;

    while (currX && currX->data != x) {
        prevX = currX;
        currX = currX->next;
    }
    while (currY && currY->data != y) {
        prevY = currY;
        currY = currY->next;
    }

    if (!currX || !currY) {
        cout << "One or both values not found.\n";
        return;
    }

    if (prevX)
        prevX->next = currY;
    else
        head = currY;

    if (prevY)
        prevY->next = currX;
    else
        head = currX;

    Node* temp = currX->next;
    currX->next = currY->next;
    currY->next = temp;

    cout << "Swapped " << x << " and " << y << ".\n";
}
```

Explanation: adjust pointer carefully

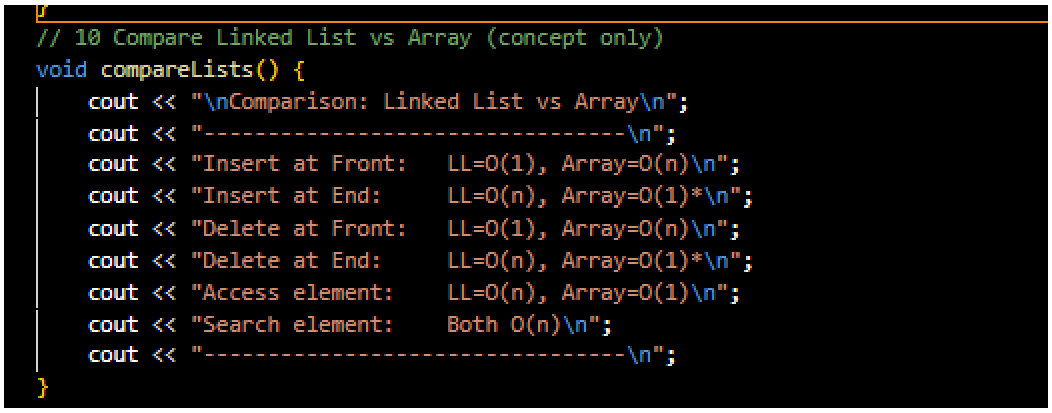
Challenge 9 – search



```
// 9 Search
void search(Node* head, int key) {
    int pos = 1;
    while (head) {
        if (head->data == key) {
            cout << key << " found at position " << pos << ".\n";
            return;
        }
        head = head->next;
        pos++;
    }
    cout << key << " not found in list.\n";
}
```

Challenge 10 – compare with arrays

Explanation: link list vs array



```
// 10 Compare Linked List vs Array (concept only)
void compareLists() {
    cout << "\nComparison: Linked List vs Array\n";
    cout << "--------------------------------\n";
    cout << "Insert at Front:   LL=O(1), Array=O(n)\n";
    cout << "Insert at End:     LL=O(n), Array=O(1)*\n";
    cout << "Delete at Front:   LL=O(1), Array=O(n)\n";
    cout << "Delete at End:     LL=O(n), Array=O(1)*\n";
    cout << "Access element:    LL=O(n), Array=O(1)\n";
    cout << "Search element:    Both O(n)\n";
    cout << "--------------------------------\n";
}
```