



ChatGPT ▾

Share



ChatGPT

Hot Mods

Explore GPTs

Today

Linked List Operations

...

Yesterday

C++ Linked List Programs

Sci-Fi Group Names

Coding Student Illustration Request

DSA Program Solutions

Previous 7 Days

Pointers in C

Barcode Information Inquiry

Infinite Recursion Explanation

C++ Function Swap Correction

```
#include<bits/stdc++.h>
using namespace std;

// Structure for a node in the linked list
struct node {
    int data;      // Data stored in the node
    node* link;   // Pointer to the next node
};

node* head = nullptr; // Global head pointer initialized to nullptr
// (empty list)

// Function to insert a new node at the end of the linked list
void insert(int val) {
    // If the list is empty, create the first node
    if (head == nullptr) {
        head = new node; // Create a new node
        head->data = val; // Assign the value to the node
        head->link = nullptr; // Set the link to nullptr (end of list)
    } else {
        // If the list is not empty, traverse to the end and insert a new
        node
        node* temp = new node; // Create a new node
        temp->data = val; // Assign the value to the node
        temp->link = nullptr; // Set the link to nullptr (end of list)
        node* t = head; // Temporary pointer to traverse the list
        while (t->link != nullptr) {
            t = t->link; // Move to the next node
        }
        t->link = temp; // Link the last node to the new node
    }
}

// Function to insert a new node at a specific position
void insertAtPosition(int val, int pos) {
    node* temp = new node; // Create a new node
    temp->data = val; // Assign the value to the node
    temp->link = nullptr;

    // If inserting at the first position
    if (pos == 1) {
        temp->link = head; // Link the new node to the current head
        head = temp; // Make the new node the head
        return;
    }

    // Traverse to the position just before where we want to insert
    node* t = head;
    for (int i = 1; i < pos - 1 && t != nullptr; i++) {
        t = t->link;
    }

    // If the position is out of bounds (list is shorter than pos - 1)
    if (t == nullptr) {
        cout << "Position out of bounds" << endl;
        delete temp; // Delete the temporary node to prevent memory
        leak
    } else {
        temp->link = t->link; // Link the new node to the next node
        t->link = temp; // Link the previous node to the new node
    }
}

// Function to delete a node at a specific position
void deleteNode(int pos) {
    if (head == nullptr) {
        cout << "List is empty" << endl; // If the list is empty, print a
        message
        return;
    }

    // If deleting the first node
    if (pos == 1) {
        node* t = head; // Temporary pointer to the head
        head = t->link; // Move head to the next node
        delete t; // Delete the original head
        return;
    }

    // Traverse to the node just before the one to be deleted
    node* t = head;
```

```

node* pre = nullptr;
for (int i = 1; i < pos && t != nullptr; i++) {
    pre = t;           // Track the previous node
    t = t->link;     // Move to the next node
}

// If position is out of bounds
if (t == nullptr) {
    cout << "Position out of bounds" << endl;
} else {
    pre->link = t->link; // Link the previous node to the next node
    delete t;             // Delete the node at position pos
}
}

// Function to search for a value in the linked list
bool search(int val) {
    node* t = head; // Start from the head
    while (t != nullptr) {
        if (t->data == val) { // If the value is found, return true
            return true;
        }
        t = t->link; // Move to the next node
    }
    return false; // Return false if the value is not found
}

// Function to print the elements of the linked list
void print() {
    node* t = head; // Start from the head
    while (t != nullptr) {
        cout << t->data << " "; // Print the data
        t = t->link; // Move to the next node
    }
    cout << endl; // Print a newline at the end
}

int main() {
    // Insert elements into the linked list
    insert(10); // Insert 10 at the end
    insert(20); // Insert 20 at the end
    insert(30); // Insert 30 at the end
    cout << "List after inserting 10, 20, 30: ";
    print(); // Output: 10 20 30

    // Insert element at a specific position
    insertAtPosition(15, 2); // Insert 15 at position 2
    cout << "List after inserting 15 at position 2: ";
    print(); // Output: 10 15 20 30

    // Delete an element at a specific position
    deleteNode(3); // Delete the element at position 3
    cout << "List after deleting element at position 3: ";
    print(); // Output: 10 15 30

    // Search for an element in the linked list
    int valToSearch; // Input: 15
    if (search(valToSearch)) {
        cout << "Value " << valToSearch << " found in the list." << endl;
    } else {
        cout << "Value " << valToSearch << " not found in the list." <<
    endl;
}

    // Demonstrate inserting another element
    insert(40); // Insert 40 at the end
    cout << "List after inserting 40: ";
    print(); // Output: 10 15 30 40

    return 0;
}

```

sample input sample output



Sample Input:

The program doesn't require specific input but uses functions inside `main()` to manipulate the linked list. Here's a breakdown of the operations performed:

1. Inserting values: 10, 20, 30
2. Inserting a value at position 2: 15
3. Deleting a node at position 3
4. Searching for value 15

5. Inserting another value at the end: 40

Sample Output:

mathematica

Copy code

```
List after inserting 10, 20, 30: 10 20 30
List after inserting 15 at position 2: 10 15 20 30
List after deleting element at position 3: 10 15 30
Value 15 found in the list.
List after inserting 40: 10 15 30 40
```

Explanation:

1. After inserting 10, 20, 30: The list becomes 10 20 30.
2. After inserting 15 at position 2: The list becomes 10 15 20 30.
3. After deleting the element at position 3: ↓ list becomes 10 15 30.
4. Searching for value 15: The program finds it and prints Value 15 found in the list.
5. After inserting 40: The list becomes 10 15 30 40.

This sequence demonstrates basic linked list operations like insertion, deletion, and search.

again code

Sure! Here's the code again:

cpp

Copy code

```
#include<bits/stdc++.h>
using namespace std;

// Structure for a node in the linked list
struct node {
    int data;           // Data stored in the node
    node* link;        // Pointer to the next node
};

node* head = nullptr; // Global head pointer initialized to nullptr (empty list)

// Function to insert a new node at the end of the linked list
void insert(int val) {
    // If the list is empty, create the first node
    if (head == nullptr) {
        head = new node;           // Create a new node
        head->data = val;         // Assign the value to the node
        head->link = nullptr;     // Set the link to nullptr (end of list)
    } else {
        // If the list is not empty, traverse to the end and insert a new node
        node* temp = new node; // Create a new node
        temp->data = val;      // Assign the value to the node
        temp->link = nullptr; // Set the link to nullptr (end of list)
        node* t = head;        // Temporary pointer to traverse the list
        while (t->link != nullptr) {
            t = t->link;       // Move to the next node
        }
        t->link = temp;        // Link the last node to the new node
    }
}

// Function to insert a new node at a specific position
void insertAtPosition(int val, int pos) {
    node* temp = new node; // Create a new node
    temp->data = val;     // Assign the value to the node
    temp->link = nullptr;

    // If inserting at the first position
    if (pos == 1) {
        temp->link = head; // Link the new node to the current head
        head = temp;        // Make the new node the head
        return;
    }

    // Traverse to the position just before where we want to insert
    node* t = head;
    for (int i = 1; i < pos - 1 && t != nullptr; i++) {
        t = t->link;
    }

    // If the position is out of bounds (list is shorter than pos - 1)
    if (t == nullptr) {
        cout << "Position out of bounds" << endl;
    } else {
        temp->link = t->link; // Link the new node to the next node
        t->link = temp;        // Link the previous node to the new node
    }
}
```

```

        delete temp; // Delete the temporary node to prevent memory leak
    } else {
        ↓
        temp->link = t->link; // Link the new node to the next node
        t->link = temp;      // Link the previous node to the new node
    }
}

// Function to delete a node at a specific position
void deleteNode(int pos) {
    if (head == nullptr) {
        cout << "List is empty" << endl; // If the list is empty, print a message
        return;
    }

    // If deleting the first node
    if (pos == 1) {
        node* t = head;           // Temporary pointer to the head
        head = t->link;         // Move head to the next node
        delete t;                // Delete the original head
        return;
    }

    // Traverse to the node just before the one to be deleted
    node* t = head;
    node* pre = nullptr;          ↓
    for (int i = 1; i < pos && t != nullptr; i++) {
        pre = t;                  // Track the previous node
        t = t->link;             // Move to the next node
    }

    // If position is out of bounds
    if (t == nullptr) {
        cout << "Position out of bounds" << endl;
    } else {
        pre->link = t->link;   // Link the previous node to the next node
        delete t;                // Delete the node at position 'pos'
    }
}

// Function to search for a value in the linked list
bool search(int val) {
    node* t = head; // Start from the head
    while (t != nullptr) {
        if (t->data == val) { // If the value is found, return true
            return true;
        }
        t = t->link; // Move to the next node
    }
    return false; // Return false if the value is not found
}

// Function to print the elements of the linked list
void print() {
    node* t = head; // Start from the head
    while (t != nullptr) {
        cout << t->data << " "; // Print the data
        t = t->link; // Move to the next node
    }
    cout << endl; // Print a newline at the end
}

int main() {
    // Insert elements into the linked list
    insert(10); // Insert 10 at the end
    insert(20); // Insert 20 at the end
    insert(30); // Insert 30 at the end
    cout << "List after inserting 10, 20, 30: ";
    print(); // Output: 10 20 30

    // Insert element at a specific position
    insertAtPosition(15, 2); // Insert 15 at position 2
    cout << "List after inserting 15 at position 2: ";
    print(); // Output: 10 15 20 30

    // Delete an element at a specific position
    deleteNode(3); // Delete the element at position 3
    cout << "List after deleting element at position 3: ";
    print(); // Output: 10 15 30

    // Search for an element in the linked list
    int valToSearch = 15;
    if (search(valToSearch)) {
        cout << "Value " << valToSearch << " found in the list." << endl;
    } else {
}

```

```
        cout << "Value " << valToSearch << " not found in the list." << endl;
    }

    // Demonstrate inserting another element
    insert(40); // Insert 40 at the end
    cout << "List after inserting 40: ";
    print(); // Output: 10 15 30 40

    return 0;
}
```

This code implements basic operations on a singly linked list in C++: inserting nodes at the end or at specific positions, deleting nodes, searching for a value, and printing the list.

⌄ ⌂ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉

C++ Function Swap Correction

⊕ Upgrade plan
More access to the best models

ⓘ Message ChatGPT



ChatGPT can make mistakes. Check important info.

?