Linked List Node Structure

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

// Node structure for Linked List
struct Node {
    int data;
    Node* next;
};

// Head of the linked list (initially empty)
Node* head = nullptr;
```

1. Push Operation (Insert element at the top of the list)

This function adds an element to the top of the stack (beginning of the linked list).

```
void push(int value) {
    // Create a new node
    Node* newNode = new Node();
    newNode->data = value;

    // New node's next is the current head
    newNode->next = head;

    // Move the head pointer to the new node
    head = newNode;

    cout << value << " pushed onto the stack." << endl;
}
```

2. Pop Operation (Remove the element from the top of the list)

This function removes the top element from the stack (the first element in the linked list).

```
void pop() {
    if (head == nullptr) {
        cout << "Stack is empty, nothing to pop." << endl;
        return;
    }

    // Store the current head node
    Node* temp = head;

    // Move head to the next node
    head = head->next;
```

```
    cout << temp->data << " popped from the stack." << endl;

    // Free the old head node
    delete temp;
}
```

3. GetTop Operation (Get the value at the top of the list)

This function retrieves the value of the top element without removing it.

```
void getTop() {
    if (head == nullptr) {
        cout << "Stack is empty." << endl;
        return;
    }
    cout << "Top of the stack: " << head->data << endl;
}
```

Main Function to Illustrate Operations

```
int main() {
    push(10);
    push(20);
    push(30);  // Stack: 30 -> 20 -> 10

    getTop();  // Should display 30

    pop();     // Removes 30; Stack: 20 -> 10
    getTop();  // Should display 20

    pop();     // Removes 20; Stack: 10
    getTop();  // Should display 10

    pop();     // Removes 10; Stack is empty
    getTop();  // Should indicate the stack is empty

    pop();     // Stack is already empty, should handle the case

    return 0;
}
```

Explanation

- **Push**: Each new node is inserted at the beginning, making it the new head of the list.
- **Pop**: The head of the list (top of the stack) is removed, and the second node becomes the new head.
- **GetTop**: This simply returns the value of the node at the head without removing it.

Example Output

```
10 pushed onto the stack.
20 pushed onto the stack.
30 pushed onto the stack.
Top of the stack: 30
30 popped from the stack.
Top of the stack: 20
20 popped from the stack.
Top of the stack: 10
10 popped from the stack.
Stack is empty.
Stack is empty, nothing to pop.
```

Full Programs

Program 1: Push Operation

This program demonstrates the `push` operation, where a new node is added to the beginning of a linked list, simulating a stack push.

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

// Node structure for Linked List
struct Node {
    int data;        // To store the value of the node
    Node* next;      // Pointer to the next node in the list
};

// Head pointer for the linked list (initially empty)
Node* head = nullptr;

// Function to push a new element to the top of the stack
void push(int value) {
    // Step 1: Create a new node
    Node* newNode = new Node();     // Allocate memory for a new node
    newNode->data = value;          // Assign value to the new node
    cout << "Created new node with value: " << value << endl;

    // Step 2: Set the new node's next pointer to point to the current head
    newNode->next = head;           // The current head is now the second
element
    cout << "New node's next set to point to the current head." << endl;

    // Step 3: Update the head pointer to point to the new node
    head = newNode;                 // New node becomes the head
    cout << value << " pushed onto the stack. Head updated to the new node."
<< endl;
}
```

```
// Function to display the current stack
void display() {
    Node* temp = head;              // Start from the head node
    cout << "Stack elements: ";
    while (temp != nullptr) {       // Traverse the list until the end
        cout << temp->data << " ";  // Print the value of each node
        temp = temp->next;          // Move to the next node
    }
    cout << endl;
}

int main() {
    push(10);  // Push 10 to the stack
    push(20);  // Push 20 to the stack
    push(30);  // Push 30 to the stack

    display(); // Display stack: 30 -> 20 -> 10

    return 0;
}
```

Explanation:

- **Node structure**: Each `Node` has a `data` (integer value) and a pointer `next` to the next node in the list.
- **Push function**:
    1. A new node is created using `new Node()`.
    2. The `data` of the new node is set to the value passed as an argument.
    3. The new node's `next` is set to the current `head`.
    4. The `head` pointer is updated to point to the new node.
- **Display function**: This function prints all elements currently in the linked list.

Program 2: Pop Operation

This program demonstrates the `pop` operation, where the top element of the stack (beginning of the linked list) is removed.

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

// Node structure for Linked List
struct Node {
    int data;        // To store the value of the node
    Node* next;      // Pointer to the next node in the list
};

// Head pointer for the linked list (initially empty)
Node* head = nullptr;
```

```
// Function to pop the top element from the stack
void pop() {
    // Step 1: Check if the stack is empty
    if (head == nullptr) {                 // If head is null, the stack is
empty
        cout << "Stack is empty, nothing to pop." << endl;
        return;
    }

    // Step 2: Store the current head node in a temporary pointer
    Node* temp = head;                     // Store current head in temp

    // Step 3: Move the head to the next node
    head = head->next;                     // Update head to point to the
next node
    cout << temp->data << " popped from the stack." << endl;

    // Step 4: Free the old head node
    delete temp;                           // Delete the node that was
removed
}

// Function to push a new element to the stack (to test pop operation)
void push(int value) {
    Node* newNode = new Node();   // Create a new node
    newNode->data = value;        // Set the value
    newNode->next = head;         // Set next to current head
    head = newNode;               // Update head to new node
    cout << value << " pushed onto the stack." << endl;
}

// Function to display the current stack
void display() {
    Node* temp = head;            // Start from the head node
    cout << "Stack elements: ";
    while (temp != nullptr) {     // Traverse the list until the end
        cout << temp->data << " "; // Print the value of each node
        temp = temp->next;        // Move to the next node
    }
    cout << endl;
}

int main() {
    // Push a few elements to the stack
    push(10);
    push(20);
    push(30);  // Stack: 30 -> 20 -> 10

    display(); // Display the current stack

    pop();     // Pop the top element (30); Stack: 20 -> 10
    display(); // Display the stack after pop

    pop();     // Pop the top element (20); Stack: 10
    display(); // Display the stack after pop
```

```
    pop();     // Pop the top element (10); Stack: empty
    display(); // Display the stack after pop

    pop();     // Attempt to pop from an empty stack

    return 0;
}
```

Explanation:

- **Pop function**:
    1. It first checks if the stack (linked list) is empty by checking if `head == nullptr`.
    2. If not empty, it stores the current `head` node in a temporary variable.
    3. The `head` pointer is updated to point to the next node (`head = head->next`).
    4. The node that was previously at the top is deleted using `delete`.
- **Push and Display functions**: These are used to test the `pop` operation and visualize the stack before and after each operation.

Sample Output for Both Programs:

Push Operation Program:

```
Created new node with value: 10
New node's next set to point to the current head.
10 pushed onto the stack. Head updated to the new node.
Created new node with value: 20
New node's next set to point to the current head.
20 pushed onto the stack. Head updated to the new node.
Created new node with value: 30
New node's next set to point to the current head.
30 pushed onto the stack. Head updated to the new node.
Stack elements: 30 20 10
```

Pop Operation Program:

```
10 pushed onto the stack.
20 pushed onto the stack.
30 pushed onto the stack.
Stack elements: 30 20 10
30 popped from the stack.
Stack elements: 20 10
20 popped from the stack.
Stack elements: 10
10 popped from the stack.
Stack elements:
Stack is empty, nothing to pop.
```

Program 3: Push Operation (Generic Version)

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

// Node structure for a generic Linked List using templates
template <typename T>
struct Node {
    T data;          // To store the value of any type (int, float, char,
etc.)
    Node* next;      // Pointer to the next node in the list
};

// Head pointer for the linked list (initially empty)
template <typename T>
Node<T>* head = nullptr;

// Function to push a new element of any type to the top of the stack
template <typename T>
void push(T value) {
    // Step 1: Create a new node
    Node<T>* newNode = new Node<T>();   // Allocate memory for a new node
    newNode->data = value;              // Assign the value to the new node
    cout << "Created new node with value: " << value << endl;

    // Step 2: Set the new node's next pointer to point to the current head
    newNode->next = head<T>;            // The current head is now the second
element
    cout << "New node's next set to point to the current head." << endl;

    // Step 3: Update the head pointer to point to the new node
    head<T> = newNode;                  // New node becomes the head
    cout << value << " pushed onto the stack. Head updated to the new node."
<< endl;
}

// Function to display the current stack
template <typename T>
void display() {
    Node<T>* temp = head<T>;             // Start from the head node
    cout << "Stack elements: ";
    while (temp != nullptr) {            // Traverse the list until the end
        cout << temp->data << " ";       // Print the value of each node
        temp = temp->next;               // Move to the next node
    }
    cout << endl;
}

int main() {
    // Push different types of data into the stack
    push<int>(10);           // Push an integer
    push<float>(20.5f);      // Push a float
    push<char>('A');         // Push a character
    push<double>(30.99);     // Push a double
```

```
    display<int>();          // Display stack for integers
    display<float>();        // Display stack for floats
    display<char>();         // Display stack for chars
    display<double>();       // Display stack for doubles

    return 0;
}
```

Explanation:

- **Template definition**: We define the node structure and functions using the template `template <typename T>`, where `T` can be any data type (`int`, `float`, `char`, `double`, etc.).
- **Push function**: The function works with any data type. We use `Node<T>` to ensure the node can store any type of data.
- **Display function**: This function also works generically for any data type.
- In the `main` function, we push and display different types of data (`int`, `float`, `char`, and `double`).

Program 4: Pop Operation (Generic Version)

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

// Node structure for a generic Linked List using templates
template <typename T>
struct Node {
    T data;          // To store the value of any type (int, float, char,
etc.)
    Node* next;      // Pointer to the next node in the list
};

// Head pointer for the linked list (initially empty)
template <typename T>
Node<T>* head = nullptr;

// Function to pop the top element of any type from the stack
template <typename T>
void pop() {
    // Step 1: Check if the stack is empty
    if (head<T> == nullptr) {            // If head is null, the stack is
empty
        cout << "Stack is empty, nothing to pop." << endl;
        return;
    }

    // Step 2: Store the current head node in a temporary pointer
    Node<T>* temp = head<T>;             // Store current head in temp
```

```
    // Step 3: Move the head to the next node
    head<T> = head<T>->next;             // Update head to point to the next
node
    cout << temp->data << " popped from the stack." << endl;

    // Step 4: Free the old head node
    delete temp;                         // Delete the node that was removed
}

// Function to push a new element of any type to the stack
template <typename T>
void push(T value) {
    Node<T>* newNode = new Node<T>();    // Create a new node
    newNode->data = value;               // Set the value
    newNode->next = head<T>;             // Set next to current head
    head<T> = newNode;                   // Update head to new node
    cout << value << " pushed onto the stack." << endl;
}

// Function to display the current stack
template <typename T>
void display() {
    Node<T>* temp = head<T>;             // Start from the head node
    cout << "Stack elements: ";
    while (temp != nullptr) {            // Traverse the list until the end
        cout << temp->data << " ";       // Print the value of each node
        temp = temp->next;               // Move to the next node
    }
    cout << endl;
}

int main() {
    // Push different types of data into the stack
    push<int>(10);
    push<float>(20.5f);
    push<char>('A');
    push<double>(30.99);

    display<int>();          // Display stack for integers
    display<float>();        // Display stack for floats
    display<char>();         // Display stack for chars
    display<double>();       // Display stack for doubles

    // Pop different types of data from the stack
    pop<int>();              // Pop an integer
    pop<float>();            // Pop a float
    pop<char>();             // Pop a char
    pop<double>();           // Pop a double

    return 0;
}
```

Explanation:

- **Pop function**: Like the `push` function, the `pop` function is also generic and works with any data type.
- **Template usage**: In the `main` function, we demonstrate `pop` and `push` for different types of data (`int`, `float`, `char`, and `double`).

Sample Output for Both Programs:

Push Operation Program:

```
Created new node with value: 10
New node's next set to point to the current head.
10 pushed onto the stack. Head updated to the new node.
Created new node with value: 20.5
New node's next set to point to the current head.
20.5 pushed onto the stack. Head updated to the new node.
Created new node with value: A
New node's next set to point to the current head.
A pushed onto the stack. Head updated to the new node.
Created new node with value: 30.99
New node's next set to point to the current head.
30.99 pushed onto the stack. Head updated to the new node.
Stack elements: 30.99
Stack elements: 20.5
Stack elements: A
Stack elements: 10
```

Pop Operation Program:

```
10 pushed onto the stack.
20.5 pushed onto the stack.
A pushed onto the stack.
30.99 pushed onto the stack.
Stack elements: 30.99
Stack elements: 20.5
Stack elements: A
Stack elements: 10
30.99 popped from the stack.
20.5 popped from the stack.
A popped from the stack.
10 popped from the stack.
```