Part 3 - Searching

1. Searching in a Custom Singly Linked List

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

// Linked list node
class Node
{
public:
    int key;
    Node* next;
};

// Add a new node at the front
void push(Node** head_ref, int new_key)
{
    Node* new_node = new Node();
    new_node->key = new_key;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

// Search for a value in the linked list
bool search(Node* head, int x)
{
    Node* current = head;
    while (current != NULL)
    {
        if (current->key == x)
            return true;
        current = current->next;
    }
    return false;
}
```

```
int main()
{
    Node* head = NULL;
    int x = 21;

    // Construct list: 14->21->11->30->10
    push(&head, 10);
    push(&head, 30);
    push(&head, 11);
    push(&head, 21);
    push(&head, 14);

    search(head, 21) ? cout << "Yes" : cout << "No";
    return 0;
}
```

Explanation:

- `Node` class represents each node in the linked list.
- `push` adds a new node at the beginning.
- `search` traverses the list iteratively to find the key.

2. Searching in STL `std::list`

C++ Standard Template Library (STL) provides the `list` container which can be used to implement a linked list. Searching can be done using the `std::find` algorithm.

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

int main() {
    list<int> l = {14, 21, 11, 30, 10};
    int x = 21;

    auto it = find(l.begin(), l.end(), x);

    if (it != l.end())
        cout << "Yes";
    else
        cout << "No";
```

```
    return 0;
}
```

Explanation:

- `std::list` is a doubly linked list implementation.
- `std::find` iterates through the list to locate the element.
- Returns an iterator to the element if found, or `l.end()` if not found.

Revision #2
Created 2025-09-02 08:47:35 UTC by BH
Updated 2025-09-02 08:51:14 UTC by BH