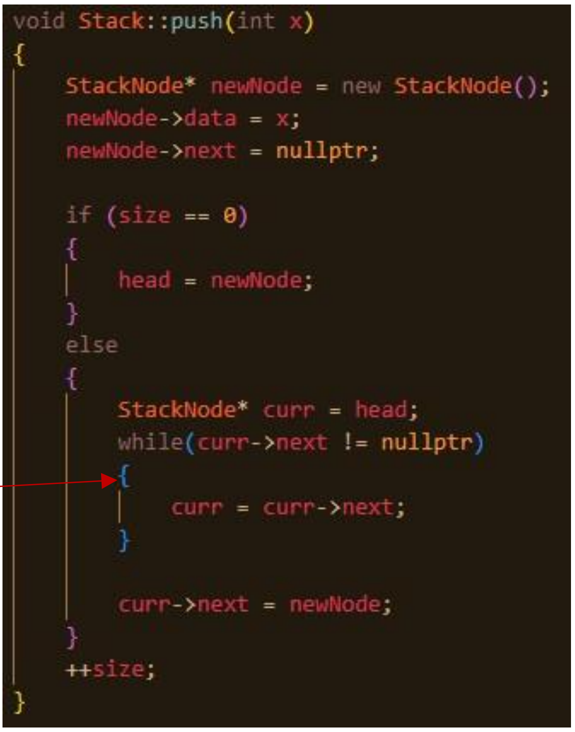
Big-O Notation Analysis

Push (): The push function requires O(n) time to push an element onto the stack. When a new item is added, we use a temporary node to traverse the linked list until the end, then we insert the new node in the n + 1 position. This requires n operations as each traversal is one operation leading to a O(n) time complexity. The push() function's time complexity would linearly increase in relation to the size of the stack.



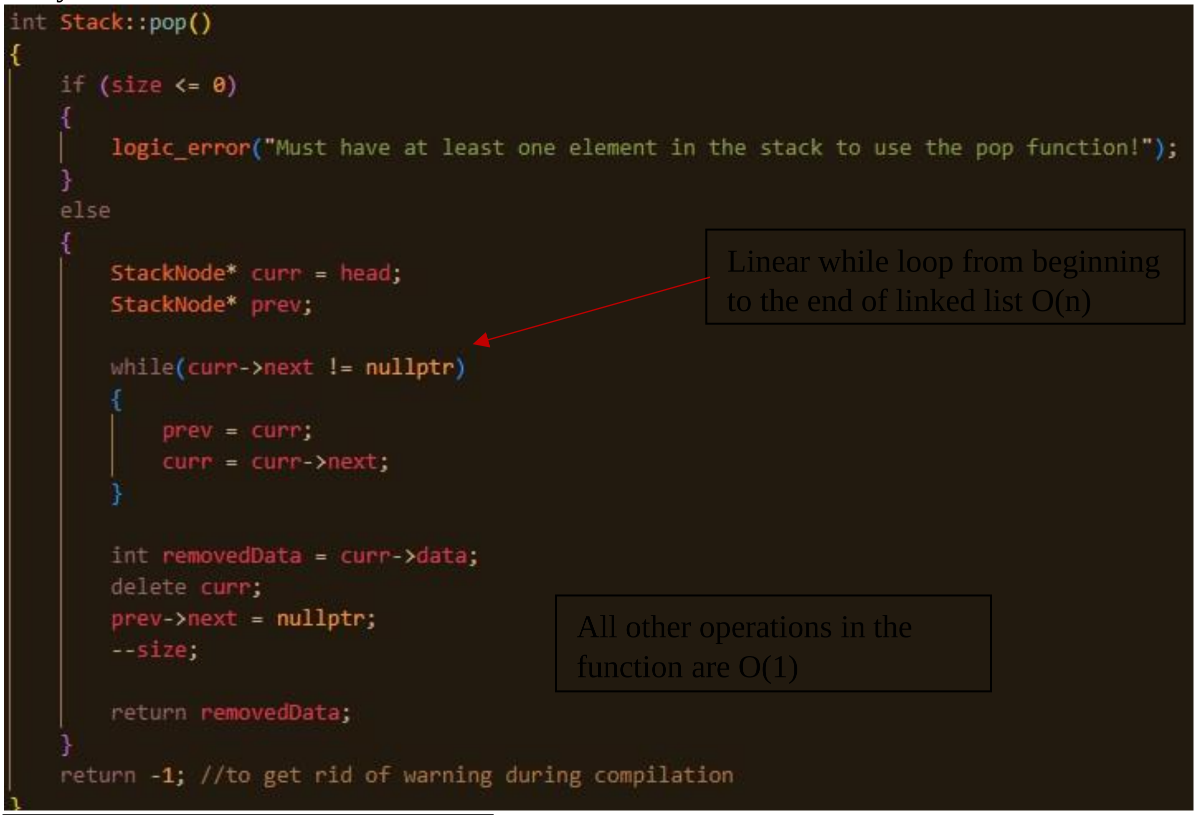
```
void Stack::push(int x)
{
    StackNode* newNode = new StackNode();
    newNode->data = x;
    newNode->next = nullptr;

    if (size == 0)
    {
        head = newNode;
    }
    else
    {
        StackNode* curr = head;
        while(curr->next != nullptr)
        {
            curr = curr->next;
        }

        curr->next = newNode;
    }
    ++size;
}
```

Total time complexity: 2n + 8

Linear while loop from beginning to the end of linked list O(n)

All other operations in the function are O(1)

Pop (): Similar to the push function, the pop function also runs on O(n) time complexity. When an element is popped off the stack all we are still required to traverse the entire loop to get to the "popable" (top most) element. The time complexity for the pop() function would increase linearly in relation to the size of the stack.



```
int Stack::pop()
{
    if (size <= 0)
    {
        logic_error("Must have at least one element in the stack to use the pop function!");
    }
    else
    {
        StackNode* curr = head;
        StackNode* prev;

        while(curr->next != nullptr)
        {
            prev = curr;
            curr = curr->next;
        }

        int removedData = curr->data;
        delete curr;
        prev->next = nullptr;
        --size;

        return removedData;
    }
    return -1; //to get rid of warning during compilation
}
```

Linear while loop from beginning to the end of linked list O(n)

All other operations in the function are O(1)

Total time complexity: 3n + 8