**Stack:**

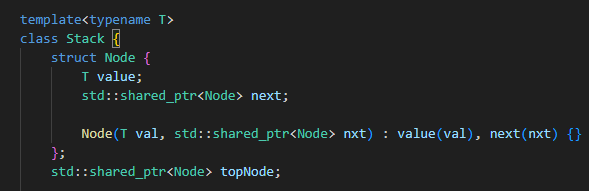
**For C++:**

**Template<typename T>:** to store any type of element, the Stack class became type-independent.

The Stack class implements a persistent stack using a linked list structure with std::shared\_ptr for memory management.

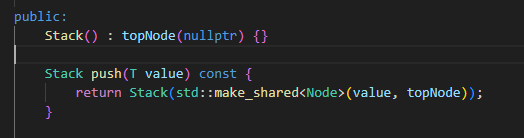
**The Stack consists of:**

* **Node**: Represents an individual element in the stack with a **value** and a **pointer to the next node.**
* **topNode:** Points to the top node of the stack.

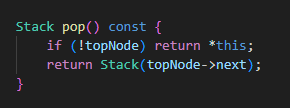


**Methods of the class :** push/pop/calculate memory size.

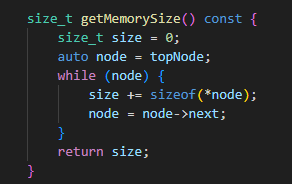
* **Push:** Creates a new node with the given value and links it to the existing stack



* **Pop:** Returns a new stack by removing the top node

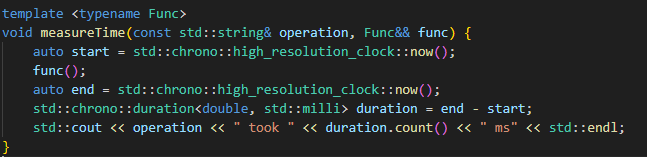


* **Memory Size Calculation**: Traverses the stack and sums the memory occupied by each node

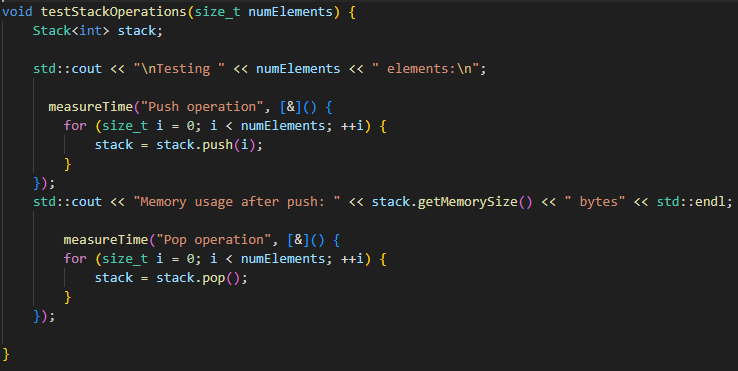


**Template<typename funct>:** to measure the time taken for any callable object, the measureTime function becomes generic.

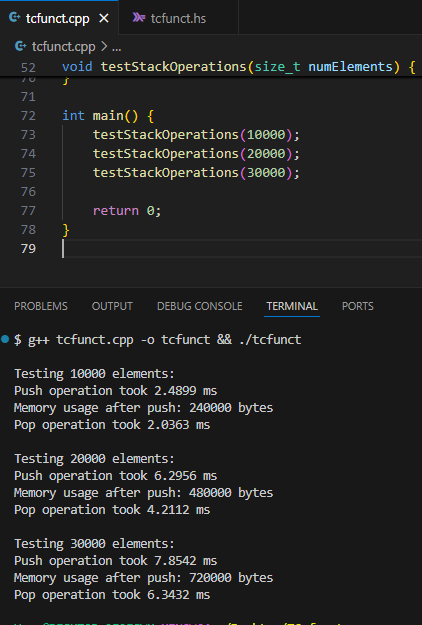
* The **measureTime** function calculates the time taken for an operation using std::chrono



* The **testStackOperations** function tests the stack for a specified number of elements (numElements):
* Push: Pushes numElements into the stack and calculates the memory usage after the operation
* Pop : Pops numElements from the stack



**Execution:**



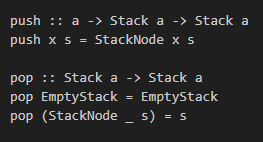
**For Haskell:**

**Persistent class:**

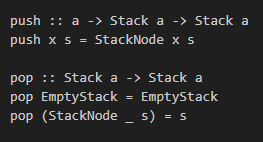
The Stack data type is a recursive structure



* **Push:** Adds an element to the stack

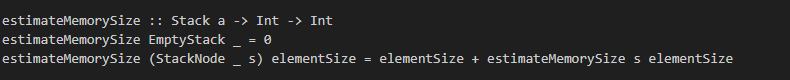


* **Pop:** Removes the top element from the stack

****

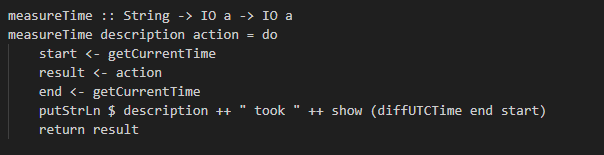
**Memory usage:**

Memory usage is approximated using the size of each element and the stack's length.



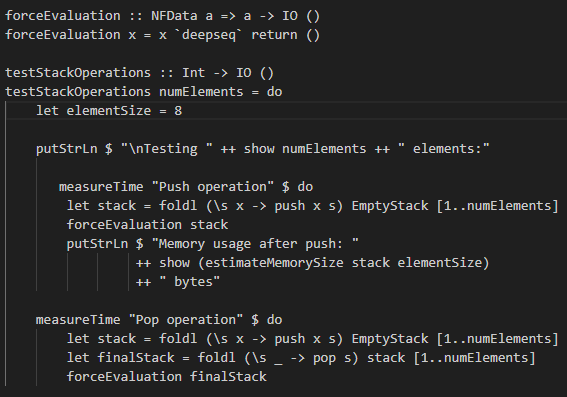
**Performance measuring:**

The **measureTime** function calculates the time taken for an operation.

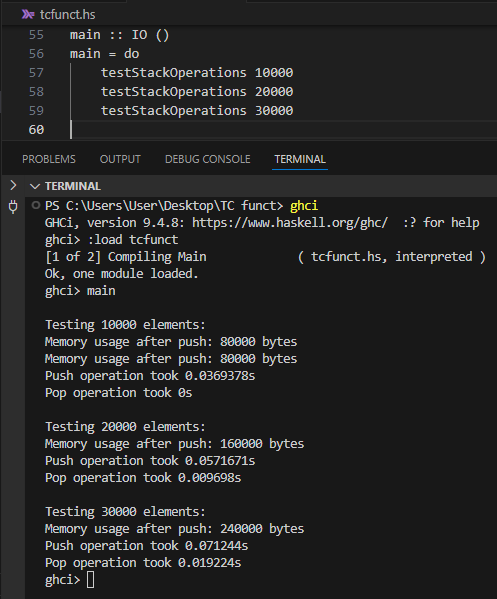


**Testing function:**

The **testStackOperations** function performs push and pop operations for a specified number of elements. (push/pop)



**Execution:**



**Comparison:**

**Time:**

* **C++ is faster** than Haskell for both push and pop operations, primarily due to the lower overhead of C++'s compiled nature and efficient memory management with std::shared\_ptr.
* Haskell's slightly slower performance is attributed to the immutable nature of functional programming and the use of recursive structures.

**Memory Usage**:

* Both implementations scale linearly with the number of elements.
* Haskell has slightly higher memory usage due to additional overhead from lazy evaluation and list-like structures.

**Carl BASSOUS 221041**

**Sophia FREIJI 222312**