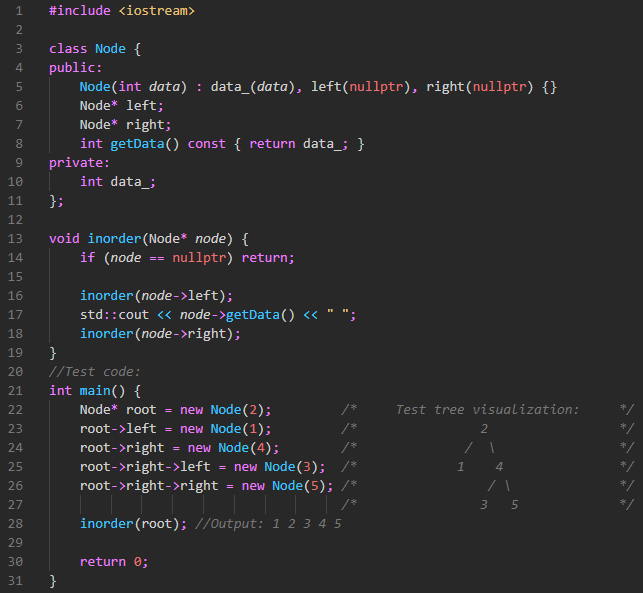
SPL Final Project: Code Examples

*Lua vs. C++*

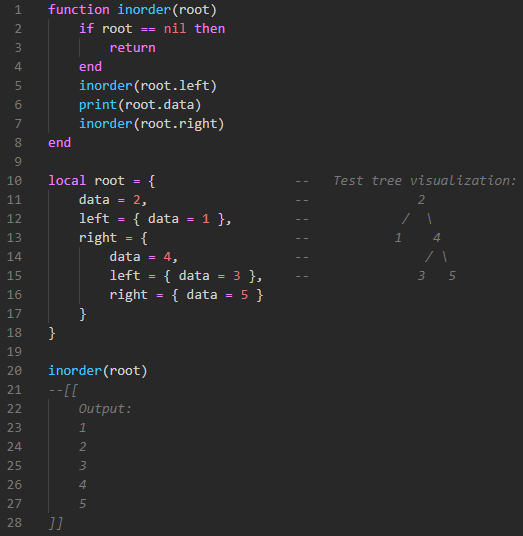
1. **Object-Oriented vs. Procedural**

This is perhaps the biggest difference between C++ and Lua: C++ is object-oriented, whereas Lua is purely procedural. Consider the following implementation of the infix traversal of a binary tree. With C++, we can use a Node class to represent the tree, but in Lua, we have to represent the entire tree with Lua’s Table data structure, which are essentially associative arrays; Lua does not support OOP.

C++:



Lua:

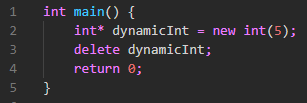


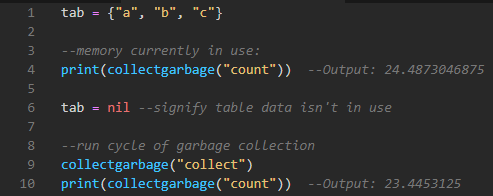
1. **Garbage Collection**

C++ does not have a garbage collector, any dynamically allocated data must be deallocated at the end of the program, or else there will be a memory leak. Lua, on the other hand, uses an incremental mark and scan strategy. Incremental means that it interleaves garbage collection with program execution, and mark-and-scan is an algorithm that utilizes a mark phase and a scan phase to group adjacent memory together to be collected.

Lua has some functions that allow us to manually demonstrate the garbage collection (thanks to TutorialsPoint for the example). So, while it’s normally done automatically, we can show it in an example below.

C++:

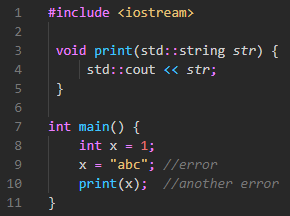


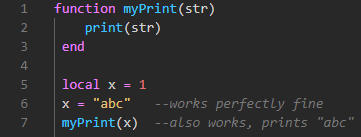
Lua:

1. **Types**

The biggest difference here is that C++ is statically typed, whereas Lua is dynamically typed. In C++, types are defined at compile time and cannot be changed. If a function expects a specific type, a mismatch will cause an error. However, for dynamically typed languages like Lua, this isn’t the case. According to lua.org, “variables do not have types; only values do. There are no type definitions in the language. All values carry their own type.” This means that you can freely reassign a variable to whatever type you wish.

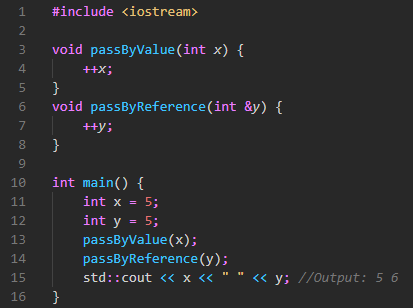
C++:

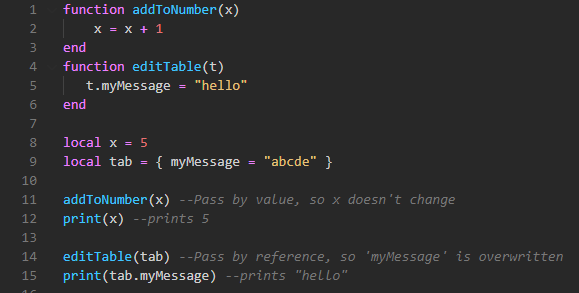


Lua:

1. **Parameter passing methods**

C++ supports both pass-by-value and pass-by-reference parameter passing. Programmers can signify they’re using pass-by-reference with an ampersand (&), however arrays are always pass-by-reference. In Lua, the programmer doesn’t have such power. Numbers and Booleans, for example, are always passed by value; they’re copied. However, even though Lua is procedural, some types are represented internally as objects. These types use pass-by-reference. For example, tables and functions (which are their own type in Lua) are passed by reference.

C++:

Lua:

**References outside of class materials**

<http://www.lua.org/manual/5.1/manual.html#2.2>

<https://www.tutorialspoint.com/lua/lua_garbage_collection.htm>