Matthew Lonis

Jeff Whitmer

CSCI-A 290

September 7th, 2016

MINI 2

**Strongly-typed versus Weakly-typed Programming Languages**

URLs

1. <http://stackoverflow.com/questions/2690544/what-is-the-difference-between-a-strongly-typed-language-and-a-statically-typed>
2. <http://stackoverflow.com/questions/17072179/difference-between-strongly-and-weakly-typed-languages>
3. <http://stackoverflow.com/questions/2351190/static-dynamic-vs-strong-weak>

Summaries

URL 1

This URL is a forum post from stack overflow however I found it to be the most reliable website for information regarding this topic. Two posts on this page made an immediate point to define two terms; static and dynamic typing. Static typing is where the type is bound to the variable and types are checked at compile time. Dynamic typing is where the type is bound to the value and types are checked at runtime. The posts eventually touched on the topic of “strong” vs “weak” typing and basically said that these terms have no real agreed upon meaning. Strong typing is usually referred to static typing, however some refer to it as a way to imply that a programming language has a strong type discipline that is enforced by the compiler or that the type discipline is hard to subvert. Weak typing implies that the compiler does not enforce a typing discipline or has enforcement that can be easily subverted.

URL 2

This URL offers another similar set of information regarding this topic and agrees with URL 1 in that strongly typed and weakly typed languages are terms that aren’t really agreed upon. One post made an effort to say that a strongly-typed language has expressions that have types which can be determined at compile time and only operations appropriate to that type are allowed. This post also referred to weakly typed languages as untyped or dynamically typed. Weakly typed languages allow any operation to be performed on any type. An important note is that while the language may allow the operation, the runtime may not.

URL 3

This URL again agrees with the previous two URLs but offers a distinction between the terms dynamic/static and weak/strongly typed. This URL says that static/dynamic typing is about when type information is acquired. This refers to whether the type information is acquired at compile time or at run time. Strong/weak typing is about how strictly types are distinguished. For example, this terminology would refer to whether a language allows or tries to do implicit conversion from strings to numbers. This post made the point to say that the C language is a weakly typed language because the data at memory addresses can be treated as a different type by casting.

Summary

All in all, weak typing is referred to languages that determine types for data during run time whereas strongly typed languages refer to languages that check data types at compile time. The terms strong/weak typing create a lot of debate because they aren’t officially or generally agreed upon for a set definition. I learned dynamic/static typing are more accurately defined terms that others recognize. Honestly, they seem to mean the same as strong/weak.

JIT 2

Are pointers the only reason C is a weakly typed (dynamically typed) language?

**Arrays**

Hoover

Hoover states that an array is intended to group together a list of values, all of the same type, under one variable name. This is beneficial because it is easier to code computations on one array rather than on a list of independently named variables. In arrays, each value occupies a cell. Every cell in an array is of the same size and occupies the same amount of memory. The size of these cells is dictated by the data types of the cell or of the array. Cells are accessed using indices and as an import note, the first cell in any array is index zero; not index one.

URLs

1. <http://www.tutorialspoint.com/cprogramming/c_arrays.htm>
2. <http://www.cprogramming.com/tutorial/c/lesson8.html>

URL 1

This URL defines arrays in a similar way as Hoover just with different language. This URL described arrays as a collection of variables of the same primitive data type. This website talked about how to declare arrays. To declare an array you must state the *type arrayName [ arraySize ];* in order to successfully declare an array. This however is only a single-dimensional array declaration. C supports multidimensional arrays but I believe going into this is outside of the scope of this class. An array must be initialized after it is declared. This means defining the list that is the array. Declaring the array creates a reservation in the memory address for the array, initializing the array assigns values to the cells in the array.

URL 2

This URL quite literally is the same as the above URL. Both URLs touched on the same topics. This URL went a little more in depth on multidimensional arrays. Multidimensional are arrays of arrays holding multiple “dimensions” of data. This URL touched on another thing the other URL didn’t touch on and that is arrays act like pointers. That is if you have a pointer, you can assign it an array without the & symbol whereas a pointer being defined to a non-array data would require the & in the pointer initialization.

Summary

To summarize arrays are a way of holding multiple variables of the same data type without having to name each variable. Arrays often time allow for quicker, easier, and space saving computations because of this simple fact; there are less variables. Arrays also vastly allow code to be more readable.

JIT 2

Are we going to be practicing with multidimensional arrays in this class?

**Strings in C**

Hoover

A string is a specific type of array. A string is specifically an array of chars where the last value of a string is ‘\0’ in order to signal the end of the string. The array that holds the chars could be of any size but the string is only from the first cell to the first cell where ‘\0’ appears. If more chars appear after ‘\0’, the programs in C will not recognize or display these chars without creating loopholes. The value of ‘\0’ is zero. This is not the same as the whole number 0 or the real number 0.0 but is instead a way of saying zero in the ASCII symbols.

URLs

1. <http://www.tutorialspoint.com/cprogramming/c_strings.htm>
2. <http://www.cprogramming.com/tutorial/c/lesson9.html>

URL 1

Strings are a one-dimensional array of characters terminated by a null character ‘\0’. There are many built in string functions in C. The null character at the end of a string is not added by the user but is instead added by the compiler. You can add the null character yourself by explicitly stating and declaring an array as follows:

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

however this can be simplified to –

char greeting[] = "Hello";

In the second example, this is an instance where the compiler would add the null character to the end of the string without us explicitly stating so.

URL 2

This URL went very in depth on strings and built in functions. I feel this website goes beyond the scope of the class but does offer some good information. The URL warns about safe programming and t be safe when manipulating and running functions on strings to make sure the null characters are always in the right places and get check, etc.

Summary

All in all, strings are simply a one-dimensional array of strings. Strings allow for the storage and representation of ASCII characters in order to hold input from a user or to display input from the user. Strings always end with a null character (‘\0’).

JIT 2

Can there be two-dimensional strings? In other words, would a two-dimensional array of characters make sense?