Timothy Astacio

CSC344 – Programming Language

5/12/2023

Task 1: The Runtime Stack and the Heap

System level operations like memory management are topics that an entry level computer science student might not need to be concerned about. However, to be a proficient programmer an understanding of these operations is critical. In memory management there are two locales that are typically used to store data onto a computer – the stack and the heap. A stack data structure follows the first in last out format what I mean by that is only the top of the stack can be accessed at any given point. With that in mind, a stack will be executed in order of most to least recently added to the stack. The problem with storing data on a stack is that the data must be located continuously on the stack which limits the size that a stack can be. Because a stack's size is limited it's important not to clutter it and instead to only use it with local variables/objects. Alternatively, a heap is not continuous in nature and is instead a collection of different memory addresses each with values. This lengthens the process to store data on the heap though because we must be explicit about where and what we are making space for. That is the reason we store persistent data that will be referenced time and time again on the heap for that reason.

Task 2: Explicit Memory Allocation/Deallocation vs Garbage Collection

Memory management is using finite resources of memory as efficiently as possible. When you allocate memory, you must always be careful to free that same memory before using it especially if you have a complicated system and you need to allocate several small pieces of memory you must remember to free those too otherwise you will run into problems. Implicit memory allocation is memory that is allocated usually on the system stack by the compiler and explicit memory allocation occurs through pointers. If you were to be using either of these in a dynamic situation where memory is being allocated and deallocated repeatedly and the chunks must be available at unknown times for an unknown length you'll have many complications. To deal with those problems there are automatic memory manager systems better yet known as garbage collectors and they can be used to automate the process. Garbage collection is a memory recovery feature built into some programming languages such as Java and C#. Garbage collectors automatically free up memory space that has been allocated to objects no longer needed. Garbage collecting makes sure that the memory limit is not exceeded or that the program reaches a point where it can no longer function. It frees developers from the responsibility of having to manually manage the memory which reduces the chances for bugs drastically and just increases quality of life.

Task 3 Salient Sentences:

1. Rust works the same way. When we declare a variable within a block, we cannot access it after the block ends. (In a language like Python, this is not the case!)

2. C++ doesn't automatically de-allocate for us! In this example, we must delete myObject at the end of the for-loop block. We can't de-allocate it after, so it will leak memory! So, it's neat that Rust handles deletion for us.

3. First, s1 "owns" the heap memory. So, when s1 goes out of scope, it will free the memory. But declaring s2 gives over ownership of that memory to the s2 reference. So s1 is now invalid. Memory can only have one owner.

4. Like in C++, we can pass a variable by reference. We use the ampersand operator (&) for this. It allows another function to "borrow" ownership, rather than "taking" ownership.

5. If you want a mutable reference, you can do this as well. The original variable must be mutable, and then you specify mut in the type of signature. You can only have a single mutable reference to a variable at a time!

6. As a final note, if you want to make a true deep copy of an object, you should use the clone function.

7. Deep copies are often much more expensive than the programmer intends. So a performance-oriented language like Rust avoids using deep copying by default.

8. Another important thing to understand about primitive types is that we can copy them.

9. We don't need to call delete as we would in C++. We define memory cleanup for an object by declaring the drop function.

10.We declare variables within a certain scope, like a for-loop or a function definition. When that block of code ends, the variable is out of scope. We can no longer access it.

Task 4: Paper Review Secure PL Adoption and Rust

Rust is a multi-paradigm language with elements drawn from functional, imperative, and object-oriented languages. Its traits, abstract behavior are types can have in common. It is an open-source language created by Mozilla and it focuses on helping developers create fast and secure applications. Rust, though not a very popular coding language to those around my age is being shown to have quite the importance in the computer science industry. Rust was developed by Mozilla to combat memory and safety related vulnerabilities in a simple yet effective manner compared to the complex new languages. Based on the information gathered from surveys companies that have implemented Rust as or senior software devs that had worked it reported mostly positive things. Secure software development is such an important issue because we will always be striving to find new ways to protect our data so learning Rust or gaining some insight on the language would increase your chances of being hired. If you decided to not use rust than Go is a language developed by Google for the very same purpose as Rust so adding either of these skills to your repertoire is greatly encouraged. In a next set of interviews conducted we asked our participants that were learning Rust why they chose to start and most of their responses fell along the lines of “because it is marketable or a good job skill.” Don’t become confused Rust is by no means an easy language to learn. The learning curve on this language is quite tough and that’s me putting it lightly. Some people have said it takes them up to a month to write a program without resorting to unsafe blocks. Six months in to learning Rust and many reported that they weren’t too comfortable with the language. Rust however, isn’t all bad as once you figure out a problem or your code complies correctly it increases your confidence and skills as a programmer. While the initial time to design or develop a program in rust is usually longer than most programming languages due to the borrow check however many participants have stated that Rust reduced the overall development time from start to finish.