Rust Memory Management / Perspectives on Rust

Task 1 - The Runtime Stack and the Heap

In programming, there are many ways in which memory management is dealt with; runtime stack and heap.

Stack keeps track of the set of data that has been given in the program. It'll keep track of definitions as well as functions that a user would input into their code. It helps store material and preserves like a library. Unfortunately, stacks cannot deal with how to get around issues that may arise in the code; variables are not functioning right or functions that are outputting weird results or no results at all.

Heap is another thing that helps keep track of material and code. Unlike stacks, heap deals with a lot more items and functions. Additionally, when you have a bunch of stacks, they become a heap that stores all the memory of each one of those stacks. The only downside though, is heap becomes less organized due to tremendous amount of information it receives.

Task 2 - Explicit Memory Allocation/Deallocation vs Garbage Collection

There are two ways in which memory can be managed in a programming language; allocation/deallocation and garbage collection.

Garbage collection can be seen and some programming languages. The point of it is that it takes memory that isn't being used and tosses it by removing it from the memory. In some ways this can be useful because it can remove functions and variables that have no reason to be used in code, however, it does arise in which it takes a while to process its job. A good example of a program that uses garbage collection is Python.

Allocation and deallocation are another type of memory management that can be seen in programming. The point of it is that it is similar to filling up a trash can. Allocation fills up with all the memory that is given in the program and stores it, similar to throwing out trash in the trash can. Deallocation removes all the memory from it and makes it fill up with new memory, similar to how you would take the trash out after it's full and replace it with a new garbage bag. However this becomes an issue when memory gets over written by new memory and can cause old memory that was important to be erased. C as a programming language that would do this type of memory.

Task 3 - Rust: Memory Management

“Heap memory always has **one owner**, and once that owner goes out of scope, the memory gets de-allocated.”

“When we declare a variable within a block, we cannot access it after the block ends. (In a language like Python, this is actually not the case!)”

“Another important thing to understand about primitive types is that we can copy them. Since they have a fixed size, and live on the stack, copying should be inexpensive.”

“Deep copies are often much more expensive than the programmer intends.”

“Rust will call `drop` on both of them. And they will free the same memory!”

“**Memory can only have one owner.**”

“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. When it's done, the original reference will still be valid.”

“This kind of "double delete" is a big problem that can crash your program and cause security problems.”

“In general, **passing variables to a function gives up ownership**.”

“There's one big catch though! You can only have a single mutable reference to a variable at a time! Otherwise your code won't compile! This helps prevent a large category of bugs!”

Task 4 - Paper Review: Secure PL Adoption and Rust

Restless made in order to avoid using unsafe code. And in some low-level languages, the programmer can dispose of and add on memory to a program. However, there is a possibility in which memory can get messed up through the allocate and deallocate process by jumbling up memory if it's not handled properly. Since rust is more newer than most programming languages, it's issues are still being found in modern day.

To what brings people to use rust is that it has a good performance when it comes to garbage collecting in comparison to other low-level programming languages as well as the safety of the language. In other languages and whatnot, you're usually forced to choose the lesser of two evils. Rest, on the other hand, allows safety by using the word "unsafe" to the note if certain variables or functions are not safe to use. It performs better than other languages in comparison for that specific aspect.

Similar to all programming languages, rust does have its problems. In other languages, some of them have dependencies that allow it to use libraries that can help portray handmade functions. Rust runs into the problem of not having that ability to be able to add libraries. Additionally, it is also hard to learn and adapt to. Learning it takes vigorous and steep hoops to jump over. It also runs into the problem of having to "compete" with other languages that are more simplistic in comparison to rust.