Rust Problem Set: Memory Management and Perspectives.

Abstract: This assignment features questions about Rust's memory management and some perspectives on rust. There are 5 tasks with the 5th task being the creation of this document so it will not be included in the lists below.

Interaction: Task 1 The Runtime Stack and the Heap

Computers utilize runtime stack and heap to execute programs. Most people don't know what these two vital pieces of the computer are much less what they do. Anyone who wants to learn more in depth about software should know what the runtime stack and heap are and what their function is.

The runtime stack is memory that is managed by the central processing unit. The stack is very small in size compared to other memory types like the heap. The stack stores variables from the functions that are made in a program when that program is executed.

The heap is also memory for storing data, but this memory storage is slower than the stack but is larger than the stack. The heap contains data also when the program is executed. One main point is that heap memory is visible through the program so one must take care when using the heap.

Interaction: Task 2 Explicit Memory Allocation/Deallocation vs Garbage Collection

After one has got quite a bit of experience programming one has to get introduced to the topic of memory allocation and deallocation management styles and garbage collection memory management style.

Allocating and deallocating memory is referred to the process in which a programmer has to manually set the memory sections that a computer will use. Likewise memory deallocation is the process in which a programmer frees up that memory so that the system can use it. Compared with Java that doesn't allow manual allocating and deallocating memory C in contrast does but an issues that can occur from the manual process of memory management are memory related bugs like a memory leak.

Garbage collection is the automatic detection of memory that is no longer in use and freeing up that memory so that the system can then utilize that memory. The drawback of garbage collection is that the programs that are written in a language like java tend to be slower than if they were written in a language that doesn't feature garbage collection.

Interaction: Task 3 Rust Memory Management

This section is pulled from the blog https://mmhaskell.com/rust/memory 10 sentences that I found interesting are:

1. This is the main concept governing Rust's memory model. Heap memory always has **one owner**, and once that owner goes out of scope, the memory gets de-allocated.
2. 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.
3. Rust works the same way.
4. Another important thing to understand about primitive types is that we can copy them.
5. For people coming from C++ or Java, there seem to be two possibilities. If copying into `s2` is a shallow copy, we would expect the sum length to be 12. If it's a deep copy, the sum should be 9. But this code won't compile at all in Rust! The reason is **ownership**.
6. In Rust, here's what would happen with the above code. Using `let s2 = s1` will do a shallow copy. So `s2` will point to the same heap memory. But at the same time, it will **invalidate** the `s1` variable. Thus when we try to push values to `s1`, we'll be using an invalid reference. This causes the compiler error.
7. **Memory can only have one owner.**
8. In general, **passing variables to a function gives up ownership**.
9. 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. In this example, the `s1` variable re-takes ownership of the memory after the function call ends.
10. You can only have a single mutable reference to a variable at a time!

Interaction: Task 4 Paper Review Secure PL Adoption and Rust

The review of the paper featuring Rust security is pretty well balanced. The authors go into detail about the advantages and disadvantages that rust offers, giving it a fair assessment. For someone who is about to go into industry I'd keep a few things in mind that the paper highlights.

First the benefits of using rust which include performance and safety in general. Overall, it seems that those who use Rust agree that rust features performance and safety which is hard to come by because usually languages can only offer one or the other. The language C will offering performance lacks safety and Java is just the reverse so Rust seems to be a nice middle ground.

Secondly the drawbacks of adopting Rust which include the difficulty in learning the language and the fact that few companies have unsafe code reviews. Learning the language takes time but Rust seems to take a lot longer to learn before one can feel comfortable using it as opposed to python which is easier to pick up and learn. The fact that few companies feature unsafe code reviews means that even though Rust prioritizes safety the programmer is also responsible for safe code so while learning and getting familiar with rust you might end up writing unsafe code for your company.