CSC344 Problem Set: Memory Management / Perspectives on Rust

<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

Learning Abstract: This assignment is focused on learning about memory management and writing my perspectives on Rust.

— —

Task 1 - The Runtime Stack and the Heap

Rust is a low-level programming language that is designed to be safe for programmers. The next two paragraphs will talk about the runtime of a stack, how it works, and the significance of it. What also will be mentioned is a description of the heap, along with how it works and the significance of it as well. This discussion is significant because Rust is a popular language and our class is meant to teach us new languages as well as how they work. The next two paragraphs will give us a greater understanding of how Rust functions.

A runtime stack keeps track of everything happening on a running program. The main method is executed and then added to the stack. Stack allocation is very fast but that comes with pros and cons, we can get rid of memory very fast which helps it be fast, but we can't keep values around if we need them for longer than a single function. This is why Rust does not need a garbage collector. This stack can't hold a large set of data because the stacks on a computer is limited. This is when a heap comes in.

A heap is also limited but it is much larger in size compared to a stack. It is used for dynamic memory allocation and is slow. The heap can be allocated and freed in any order. The elements inside of the heap do not depend on each other. When memory is on the heap, it can stay alive longer than the function which allocates the box.

— —

Task 2 - Explicit Memory Allocation/Deallocation vs Garbage Collection

We know that memory is important for programs and how they work. The next two paragraphs will be explaining what memory allocation and memory deallocation is, and how garbage collection works in Rust. It is important to understand how these things work in computer science because it is something that is used frequently and could be used in different ways.

Memory allocation is the process of setting aside sections of memory in a program to be used to store variables, and instances of structures and classes. It is often seen in C or C++. Memory deallocation is a way to free the Random Access Memory of finished processes and allocate new ones. These things are important to maintain good performance because if memory isn't deallocated, it could exhaust the system's memory resources. But these methods could also lead to dangling pointers, crashes, and security issues if there is no space for the memory. This is why garbage collection is also important.

Garbage collection is a memory recovery feature built into programming languages. Languages that contain garbage collection often include one or more garbage collectors that automatically free up memory space that has been allocated to objects no longer need by the program. It works by reclaiming the underlying memory when the object is no longer used, and it reuses it for future allocation. A problem with it is that it effects performance and runtime because it has to regularly run through the program to check object references and clean out memory.

— —

Task 3 - Rust: Memory Management

1. In C++, we explicitly allocate memory on the heap with `new` and de-allocate it with `delete`. In Rust, we do allocate memory and de-allocate memory at specific points in our program. Thus it doesn't have garbage collection, as Haskell does. But it doesn't work quite the same way as C++.

2. In this part, we'll discuss the notion of **ownership.** 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.

3. 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.

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

5. But string literals don't give us a complete string type. They have a fixed size. So even if we declare them as mutable, we can't do certain operations like append another string. This would change how much memory they use!

6. What's cool is that once our string does go out of scope, Rust handles cleaning up the heap memory for it! We don't need to call delete as we would in C++. We define cleanup for an object by declaring the drop 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. When **s1** and **s2** go out of scope, Rust will call **drop** on both of them. And they free the same memory! This kind of "double delete" is a big problem that can crash your program and cause security problems.

9. **Memory can only have one owner…** In general, **passing variables to a function gives up ownership.**

10. We can wrap up with a couple thoughts on slices. Slices give us an immutable, fixed-size reference to a continuous part of an array. Often, we can use the string literal type **str** as a slice of an object **String**. Slices are either primitive data, stored on the stack, or they refer to another object. This means they do not have ownership and thus do not de-allocate memory when they go out of scope.

— —

Task 4 - Paper Review: Secure PL Adoption and Rust

Computer science is the process of solving complex organizational problems using technical solutions. It is important to master the task of secure software development because it is still a common issue to this day. C and C++ are languages that are not as safe as other alternatives. Rust and Go are ranked fairly high and are some of the more secure languages. Rust is an open-source systems programming language created by Mozilla, with its first stable release in 2014 and has common similarities to Java and Haskell. Rust does not use garbage collection but the way ownership works in it, is effective.

Rust is an important language to learn because it is used for a lot of software, but it is not the strongest with mobile and web applications. The programming in it can be too restrictive at points and that is because is could compromise rust's safety guarantee. On the other hand, it does have high performance and the safety aspect is extremely helpful. Rust can be hard to learn because most programmers who tried it said that it took them months to make a compiled program without looking stuff up.

The good thing about Rust is that once the code compiles, developers can be fairly confident that the code is safe and correct, so it could lead for you to become more comfortable when you finish, without worrying about as many issues. I recommend that when you try to learn rust, just try to pick the right project that can help you learn the most. Also make sure to be patient and persistent when you are coding, not just with Rust, but with any language. Try your best and work hard, the rest will follow.