Csc344 Problem Set: Memory Management / Perspectives on Rust

Task 1 - The Runtime Stack and the Heap

Rust is a lightning-fast and memory-efficient programming language that can power performance-critical applications, run on embedded devices, and easily interface with other languages. Now, in this section of the essay, we'll discuss some of the reasons why rust is so dependable, notably the runtime of a stack, how it works, and its relevance. Also an explanation of the heap, as well as how it operates and its significance, will also be provided.

The runtime stack is a particular data structure that is used during program execution to hold information about the active methods of a computer program that is being performed. The primary function of the runtime stack is to record the caller's return address of a method to be performed. What exactly is the runtime stack? It is, after all, a stack that holds method Call Frames. It's also used to display which method is being performed and which methods have called it. The runtime stack is also related to debugging: it may observe the sequence of method calls as well as the variables in the call frame. Furthermore, if our application crashes, the runtime stack helps in examining the source of the problem.

The heap is an abstraction that aids in the determination of when to generate and take out irrelevant parts of the data memory. Now the runtime stack is extremely fast and is where memory is allocated by default in Rust. However, the allocation is confined to a single function call and is limited in size. In contrast, the heap is slower and is deliberately allocated by your software. However, its size is virtually endless and it is worldwide available.

Task 2 - Explicit Memory Allocation/Deallocation vs Garbage Collection

This portion of the essay will describe what memory allocation and memory deallocation are, as well as how Rust garbage collection works. It is critical to understand how these things function since they are commonly utilized and might be employed in a variety of situations.

Essentially, the Rust allocator expects a Structure is specified for both allocation and deallocation. That is, at deallocation time, the memory allocator learns about the intended alignment of an allocation. According to the article provided to us, languages such as C and C++ use malloc to allocate memory on a heap, which gives the programmer a lot of low-level memory management control but can lead to a lot of bugs such as double free, dangling pointers, and memory leaks.

When it is no longer possible to access old memory, the application disposes of it. When memory is not connected to another, utilized piece of memory, it is no longer accessible. First, the compiler allocates memory for the program to use, then, after the program is completed, garbage collection steps in and instantly deallocates the memory used by the program. The advantage of this is totally contingent on the scenario in which someone encounters themselves.

Task 3 - Rust: Memory Management

1. Rust's memory model differs from that of Haskell. Rust, like C++, is said to provide better control over memory utilization. In C++, we actively allocate heap memory with new and release it with delete.
2. We allocate and deallocate memory in Rust at certain moments in our program. As a result, unlike Haskell, it lacks garbage collection.
3. Memory models in Rust like heap memory always have a single owner, and when that owner exits the scope, the memory is deallocated.
4. Variables in Rust are declared within a certain scope, such as a for-loop or a function definition. When that block of code is completed, the variable is no longer in scope. We no longer have access to it.
5. Because primitive types have a fixed size and exist on the stack, copying should be easy.
6. Deep copies are frequently far more costly than programmers intend. As a result, a performance-oriented language such as Rust eliminates deep copying by default.
7. Passing variables to a function, generally, gives away ownership.
8. A variable can be sent by reference in C++. For this, we employ the ampersand operator (&). It enables another function to "borrow" rather than "take" ownership. When completed, the original reference will stay in effect.
9. A variable can only have one mutable reference at a time. Otherwise, your code will fail to build. This aids in the prevention of a wide range of bugs.
10. Slices provide an immutable, fixed-size reference to an array's continuous component. As a slice of an object String, we can frequently utilize the string

literal type str. Slices might be simple data kept on the stack or a reference to another object. Because they lack ownership, they do not deallocate memory when they exit scope.

Task 4 - Paper Review: Secure PL Adoption and Rust

Rust is a programming language that was created to address an issue in programming, specifically memory allocation and how to manage it. Some programming languages, such as Python, Java, and C++, are among the most popular merely because they are pervasive and adaptable. However, emerging languages like Rust, which build on the principles of the top languages, may one day compete with these well-known and frequently used languages.

They describe in this article how they collected programmers to program on rust and analyze what they felt about utilizing it, and there were many good responses. Included are benefits for the development lifecycle and the advancement of general secure programming abilities, as well as negatives such as a lot of practice, limited library availability, and worries about the capacity to employ additional Rust programmers in the future. Their findings have implications for encouraging the use of Rust.

The most negative aspect of Rust is its massive learning curve. The majority of individuals considered Rust more difficult to learn than other languages, but when Rust code compiles, programmers can be quite certain that the code is secure and accurate, making programmers extremely comfortable with it. Although Rust may be difficult to learn, it is always a good idea for a programmer to explore different languages to see if they fit them, and who knows, maybe Rust is the programming language you have been waiting for all along.