Csc344 Problem Set: Memory Management / Perspectives on Rust

Task 1 - The Runtime Stack and the Heap

When running a program, a memory is divided into two data structures which are The Runtime Stack and the Heap. They are notable data structures because they play a vital role in many programming languages into storing and managing memory differently in the computer's RAM (Random Access Memory). For this task, the following two paragraphs will involve information about the difference between The Runtime Stack and the Heap which will enable programmers to improve their productivity into memory management.

Regarding The Runtime Stack, it is a data structure used for static memory allocation that can access memory very fast and is divided from the memory when running a program. Every time a function is called, a stack frame is pushed on the stack which holds memory used for local variables, returning address, parameters, etc. After a function has been called, the stack frame is removed from the stack that leads into the release of the memory for the next call. However, a stack overflow error will be produced if there is no stack memory.

About the Heap, it is a data structure used for dynamic memory allocation that can access memory and is also divided from the memory when running a program except the rate of accessing memory is a bit slow. Inside the Heap, elements do not depend on each other and can always be randomly accessed as well as allocating and freeing a block at any time which makes it more difficult to record any elements of the heap that are to be allocated or free at any time.

Task 2 - Explicit Memory Allocation/Deallocation vs Garbage Collection

Explicit Memory Allocation/Deallocation and Garbage Collection are two different methods of memory management which are important for programmers to decide a better memory management method in many programming languages. For this task, the following two paragraphs will encompass information about the difference between Explicit Memory Allocation/Deallocation and Garbage Collection which will allow programmers to better decide which memory management method to utilize.

Explicit Memory Allocation/Deallocation is a method of memory management which has a high complexity and great functionality that allows the programmer to allocate or deallocate memory manually. However, this method would results in an error when the new operator fails to allocate space for the memory. Furthermore, dangling pointers is likely to occur if memory is freed too early which then leads to security issues, data corruption, crashes, etc. as well as the likelihood of memory leaks if the programmer forgets to allocate or deallocate space. As a result, most modern programming were created to provide Garbage Collection.

Garbage collection is a method of automatic memory management which is built-in in Python and Java that automatically free up memory by simply looking at elements are in use and which

are not, in the heap memory. It also automatically recovers elements that are not reachable anymore. However, there can still be memory leaks whenever the last element is referred to the first element in a memory. Furthermore, non-memory resources still has to be manually clear because Garbage Collection does not clean up these resources very well.

Task 3 - Rust: Memory Management

1. “The suggestion was that Rust allows more control over memory usage, like C++.”
2. “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. “Rust works the same way. When we declare a variable within a block, we can’t access it after the block ends.”
4. “We’ve dealt with strings a little by using string literals. 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!”
5. “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 memory cleanup for an object by declaring the `drop` function.”
6. “So a performance-oriented language like Rust avoids using deep copying by default.”
7. “When `s1` and `s2` go out of scope, Rust will call `drop` on both of them. And they will free the same memory! This kind of "double delete" is a big problem that can crash your program and cause security problems.”
8. “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.”
9. “**Memory can only have one owner.**”
10. “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 study of computational system and computers which provides programmers the ability to develop software programs using many programming languages. However, there are memory safety vulnerabilities in two programming languages like C and C++ as well as no automatic memory safety. Therefore, two programming languages known as Rust and Go were developed to be low-level, fast, and memory safe as opposed to C and C++.

Rust is known to have a lack of Garbage Collection and zero-cost abstraction which makes it appropriate for environments with constrained resources. It is also well known to have good safety as it has no null pointers, high rate of concurrency and memory safety, high rate of immutability and lifetimes. The performance on Rust is also known to be on the peak as well as having less explicitly, and lack of Garbage Collection. Furthermore, Rust enables programmers to be fairly confident in their code for being safe and correct when compiled as well as improving development in the production sequence and safe development in other programming languages. Additionally, Rust has a good ecosystem which influences organizational adoption in order to provide required support for large projects.

However, Rust has a steep learning curve as most programmers took within a month to get used to it. Moreover, Rust causes dependency bloat and does not include critical libraries and infrastructure. Some programmers are concerned about Rust's maturity and maintenance as it is a new language and they were not sure about whether it would last long or not. As a result, most programmers will have a hard time considering Rust as a trustworthy and better alternative for C++.