Memory Management/ Perspectives on Rust

Abstract:

One way to think about approaches to learning is to imagine yourself in a direct learning, guided learning, or self learning scenario. All are good, and you should do your best to be comfortable with, to benefit from, all three approaches. So far, this course has been skewed towards the direct and guided modes of learning. This problem set, on the other hand, while providing some guidance, is decidedly oriented towards the self learning mode. Thus, you, along with whatever resources you can find, are expected to make some sense of a selection of ideas, and to come out of the experience knowing a little bit more about the topics than when you entered into it.

The Runtime Stack and Heap:

Stack and heap have a number of differences and similarities between them because they are basically opposites. Stack is linear, accesses local variables, and can't be resized. Heap is hierarchical and allows you to globally access variables. Runtime stack and heap are important when learning about computer programming. They're special because they are versatile and can be very useful.

Runtime stack keeps a record of whatever is being run, so you won't lose any of the code that was in the memory. Runtime stacks stay the same size no matter the number and their size leaves room for more for a lot of stacks. It's also useful and takes up even less space since it erases data after runtime. Runtime stack is very fast making it more efficient for your software so that you don't have to wait for everything to load all the time and doesn't cost you much money.

Heap is also a form of memory management that's not entirely managed through CPU and supports Dynamic memory allocation. It's slower than the runtime stack, but unlike the runtime stack it doesn't have a specific limit on memory size. You can also specifically deallocate data on heap and resize it. A downside would be that memory in heaps can become fragmented if you don't allocate the data correctly and everything has to be done manually such as freeing blocks of memory. Heap is good for advanced programmers who want things done in a specific way in regards to block of memory.

Explicit Memory Allocation/Deallocation VS Garbage Collection

Both explicit memory allocation/deallocation and garbage collection help clear up space for more useful data. Since memory is really large and there's really no limit to how much there could be, computers need to find a way to make space for new memory. This is where memory allocation and deallocation puts the unnecessary memories into another place so that there's room for new data. Garbage collection will automatically sift through the memory and delete unnecessary memory.

One of the languages that have explicit memory allocation and deallocation is C++. Since it requires manual allocation of memory and deleting any form of memory requires manual programming as well. It also costs more for memory allocation and deallocation as it takes time and memory access is harder in these regards. Memory allocation and deallocation are done in traditional algorithms such as fist fit, binary search tree and size partitioning. If you make the wrong decision in your allocation, you can accidentally leak your memory and have to detangle your pointers.

If something in your program is used and is necessary then garbage collection won't delete it from your memory. This is useful because it saves you time in comparison to memory allocation. Some languages that use garbage collection are java, python, and haskell. Garbage collection requires run-time type systems, and it has the ability to defragment memory. These features are usually typical of high level languages.

Rust: Memory Management

- "The suggestion was that Rust allows more control over memory usage, like C++. 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."
- "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."
- "If we declare a string within a block, we cannot access it after that block ends."
- "We define memory cleanup for an object by declaring the `drop` function."
- "This is a non-primitive object type that will allocate memory on the heap."
- "C++ doesn't automatically de-allocate for us! In this example, we **must delete** `myObject` at the end of the `for` loop block."
- "Deep copies are often much more expensive than the programmer intends."
- "In general, **passing variables to a function gives up ownership**."
- "Like in C++, we can pass a variable by **reference**."

- “Slices give us an immutable, fixed-size reference to a continuous part of an array.”

Paper Review: Secure PL adoption and Rust

Rust is used to protect parts of memory that are susceptible to damages. Rust is generally easy to understand as more than a quarter of people that are newly introduced would have the same amount of understanding of the program as those who’ve used it for years. Rust is useful as it supports generic modules, macro systems, tagged unions, and pattern matching. You should try rust as it is efficient and flexible in terms of the functionality and use.

Rust has variables that are its owner and when said owner gets lost, the value does as well. When said owners also want to use another’s reference value, the reference value must be borrowed. The Rust compiler enforces the borrowing rule this helps in protecting the memory. Rust can be hard to learn, but the people who understand it have done better in regards to understanding and earning other computer languages. So Rust is definitely a good language to learn as a senior computer science major graduate.

Rust provides feedback such as error messages and it’s easy to understand the mistakes one makes when the feedback tells you exactly where you went wrong. All in all, as a senior computer science major graduate, you should adopt Rust because it would help you in understanding more computer languages in the future and it helps keep memory safe. There’s no obvious downside to adopting Rust.