Abstract

One of the most important features to help facilitate reliable design in a programming language is memory management design. There are two wide-spread approaches: manual and automatic memory management, known as garbage collection (GC). Recently, a third approach which is ownership design has been fully adapted in new modern programming languages such as Rust and Swift. Rust uses ownership to eliminate high degree memory problems such as memory leak, dangling pointer, and use after free. Rust follows deterministic syntax-driven memory management depending on static ownership rules implemented and enforced by the rustc compiler. Swift also implements ownership concept in automatic reference counting (ARC). Though the ownership concept is adapted in Swift, it is not a memory-safe language because of the possibility of strong reference cycles. In this paper, we will illustrate the fundamental of ownership and the consequences of memory safety guarantees and issues related to Rust and Swift. We also conducted an experiment to compare the elapsed time binary tree allocation and deallocation in five programming languages C, C++, Java, Swift and Rust.
References

18. Ralf Jung, Jacques-Henri Jourdan, Robbert Krebbers, and Derek Dreyer. Rustbelt:


Index Terms

Computer Science
Information Sciences

Keywords

Ownership, Manual Memory Management, Garbage Collection