Garbage Collection

Adam Múdry (xmudry01), Daniel Paul (xpauld00)

November 12, 2022

Abstract

Almost all modern programming languages make use of dynamic memory allocation. This allows objects to be allocated and deallocated even if their total size was not known at the time that the program was compiled, and if their lifetime may exceed that of the subroutine activation that allocated them. A dynamically allocated object is stored in a heap, rather than on the stack. Heap allocated objects are accessed through references. Typically, a reference is a pointer to the object.

Automatic dynamic memory management via a garbage collector resolves many of issues concerning manual memory management (dangling pointers, memory leaks). Garbage collection prevents dangling pointers from being created – an object is reclaimed only when there is no pointer to it from a reachable object, which also prevents memory leaks.

There are more strategies on how we can manage dynamically allocated memory: manual management (C, C++), memory management via a "garbage collection" (Java, Python, JavaScript, etc.) or memory ownership and borrow checking (Rust).

Garbage collectors have also various strategies each with different properties such as reference counting, cycle collection, tracing, trial deletion, incremental, generational, precise, conservative, deterministic, non-deterministic, real-time collection, etc.

We will focus on comparison between ORC garbage collector (automatic reference counting with cycle collection) and Mark & Sweep garbage collector from Nim programming language.