garbage collection

- garbage collection can be called manually or automatically using the compiler or the interpreter

memory de-allocation

- memory allocation and memory de-allocation tightly coupled together and works in conjunction

introduction

- GC is a memory pattern
- GC collects unused memory blocks
- GC increase number of free memory blocks available for allocation.
- GC methods can be called at the end of the program to prevent memory leaks
- GC methods can be called at every memory free function calls
- GC methods can be used in conjunction with AST tree traversal algorithm
- GC can work in program interpretation build phase
- GC can find an AST leafs for call tree where allocation / de-allocation call can be placed
- GC can automatically put allocation / de-allocation method calls in AST
- GC and memory allocator can calculates total required memory for the program using AST
- GC and memory allocator operate required memory blocks during the program execution

lazy strategy pattern

- memory allocator allocates all the required memory at once at the program start
- memory allocator decreases the free memory pool linearly during the program run
- memory allocated objects lifetime matches the program lifetime
- memory de-allocates all the allocated memory at the end of the program at one point in time
- memory de-allocated linearly in the end of the program execution
- memory allocator and de-allocator does not injects code into the main algorithm / AST

active strategy pattern

- memory allocator allocates memory during execution of the program
- memory allocator decreases / increases the free memory pool non-linearly during the program run
- memory allocated objects lifetime not match program lifetime
- memory de-allocates memory using AST where the variables used for the last time
- memory de-allocates non-linearly during the program execution
- memory allocator and de-allocator injects code into the main algorithm / AST

AST integration

- program variables are just the marked addresses for memory allocation / de-allocation
- program variables can be referenced as 0,1,2,..,n element of a memory pool

AST variables referencing

- function calls uses arguments
- arguments of the function calls can be used to instruct AST to put function variables onto the stack
- function call can be wrapped with code for load and save variables on stack

automatic garbage collection during program execution

generally, in program without loops support it's quite easy to find in AST points for memory free. using AST you can build source code for some kind of assembly language or find a linear graph for memory allocations. in that way the program is just a set of simple instructions operates on variables. in a linear set of instructions very easy to find a point for injecting memory cleanup instructions, cause it is the last instruction where variable being used in execution timeline. in instruction flow you can inject memory de-allocation instructions right after the last instruction uses the memory variable being freed.