Memory management: memory leaks, passive garbage collection, mark and sweep:

Halt the program,

mark all allocated memory as unneeded.

Start at all pointers in the stack, and do a depth first search

Every pieces of memory visited is marked as “still needed”

When the search is done, those marked as “unneeded” are deallocated.

Trick (reverse the pointers as we do the dfs in order to use only a constant amount of additional memory)

Oracle’s approach for the Java garbage collector.

The heap is divided into sections:

eden

S0/S1

tenured memory

permanent memory

All new allocations are from eden.

Permanent memory is used for things like class objects.

When eden fills up, the program stalls to do a garbage collection.

- only memory in eden or in one of S0/S1 is explored.

One one of S0/S1 is used, the other is empty.

Everything that survives the garbage collection is placed in the empty S0/S1. Then eden and the other S0/S1 is emptied.

If an allocated piece of memory survives a number of times (ex: 8 times), it is placed into “tenured memory”.

Trick: At the start, mark all memory in tenured as “Needed”.

What happens if something in S0 is only pointed to by something in Tenured??

What happens when tenured fills up?

Now, the system stops and does a full garbage collection including everything in tenured, permanent plus eden, s0, s1.

Do we need a strictly typed language in order to do garbage collection , reference counters, tombstones, lock and key?

- the language will know everything that is a pointers

int x = ???

pointer p = (pointer)x

int y = (int)p

return y