Lecture 24: More Storage Management

CSC 131
Fall, 2006

Dynamic Languages

- Dynamic types -- associate type descriptor w/ values of variables
- Dynamic scope -- no longer need static/access link in activation record
  - look for closest activation record with vble
- Late binding costs -- more space, slower access
- Benefits - more flexibility

Pointers

- Flexibility, but major source of run-time errors.
- "Pointers have been lumped with the goto statement as a marvelous way to create impossible to understand programs."
  Kernighan & Ritchie

Problems with Pointers

- Dereferencing uninitialized or nil pointers
- Dangling pointers (recycle active memory)
  - E.g., C allows pointers into stack
  - Explicit deallocation of active memory in heap
- Garbage: Unreachable items may clog heap
- Holes in typing system may allow arbitrary ints to be used as pointers.

Heap Management

- Stack doesn't work in some circumstances
  - functions returning functions
  - dynamically allocated memory
- Heap allows dynamic allocation/deallocation of memory.
  - Manually
  - Automatically

Managing the Heap

- Heap maintained as stack of blocks of memory
- Need strategy to handle requests and returns.
  - Best fit
  - First fit
- Fragmentation is serious problem when return
- Coalesce blocks on heap
- May need to compact memory occasionally

Automating Dispose

- Garbage collection (lazy)
- Reference counting (eager):
  - Keep track of number of references to block of memory.
  - Return it when count is 0.
  - Disadvantages:
    - space and time overhead of keeping count,
    - circular structures.

Garbage Collection

- At a given point in execution of program P, memory location m is garbage if no continued execution of P from this point can access m.
- Automatic garbage collectors start with root set and search out all memory locations accessible from root set.
- Automatic garbage collectors necessarily conservative.

Mark and Sweep Collector

- Mark "alive" elements.
- Sweep through memory and reclaim garbage
- Problems:
  - Space for marks (and stack while marking)
  - Two sweeps through memory needed
- Used in Java 1.0, 1.1, but not later

Copying Collector

- Divide memory in half -- working vs. free
- When working exhausted
  - Copy live nodes from working to free
  - Swap halves
- Evaluation:
  - Only looks at live cells
  - Can be incremental
  - Needs twice as much space, but respects cache
  - Allocation very cheap!

Generational Collector

- Only try to collect recently allocated blocks
- Divide memory into two or more generations.
- Modern Java uses copying collector for youngest and older uses mark-compact scheme
  - youngest gets lots of garbage quickly
  - mark-compact doesn't move lots of older objects