Lecture 25: Managing the Heap & Commands

CSC 131
Fall, 2010

Kim Bruce

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.
  - Weak variant used in Objective C on iphone

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.

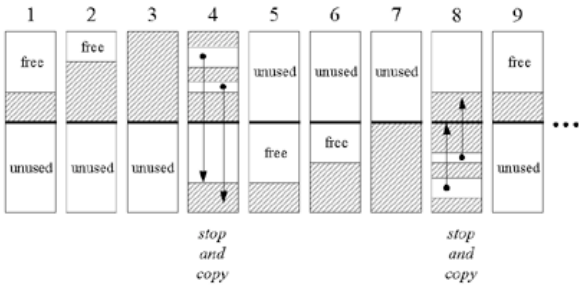
Mark and Sweep Collector

- Mark "alive" elements.
- Sweep through memory and reclaim garbage
- Problems:
  - Space for marks (and stack while marking)
  - Two sweeps through memory needed
  - Time proportional to # of live nodes
- 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 (use forwarding address)
  - Swap halves
- Evaluation:
  - Only looks at live cells, but can be incremental
  - Needs twice as much space, but respects cache
  - Allocation very cheap! Always one big block free
  - GC fast if most are dead

Copying Collector



Memory as time passes ...

Diagram from Bill Venners, Inside the Java VM

Generational Collector

- Only try to collect recently allocated blocks
  - Infant mortality - majority of blocks die young!
- 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

Implementing Parametric Polymorphism

Section 6.4 of text

Parametric Polymorphism Redux

- How do we implement polymorphic classes, functions, etc.
- Scheme, ML, Haskell, Clu (1974), Ada, C++, Eiffel, Java
- Efficient implementation depends on shared code.

C++ templates

```
template <typename T>
class Stack {
  private:
    std::vector<T> elems;     // elements

  public:
    void push(T const&);      // push element
    void pop();               // pop element
    T top() const;            // return top element
    bool empty() const {      // return if stack empty
        return elems.empty();
    }
};
```

Different T's take different amounts of space, so macro-expand at compile time

Easier if Uniform Reps

- LISP, Scheme, ML, Haskell, Clu, Eiffel, and Java have uniform reps for values so can share same code.
- Ada requires different implementation, but still type-checks statically.
- Automatic boxing and unboxing helps with primitives.

Commands

Commands

- Statements affect state
- Distinguish between state and environment
  - contents of memory
  - association btn names and values (including locations)

Assignments

- Order of evaluation
  - A[f(j)] := j * f(j) + j--
- Meaning of assignment
  - assignment by copying
  - assignment by sharing

Control Structures

- FORTRAN 1
  - GO TO n
  - GO TO (17, 43, 12, 99), I (also other variants)
  - IF(arith exp) 17, 43, 12
    means go to statement number 17 if arith exp is negative, 43 if zero, and 12 if positive
  - DO label ivble = 1, 20, 2
- Close to machine code

Goto Statements

- Why need repetition - can do it all with goto's?
- "The static structure of a program should correspond in a simple way with the dynamic structure of the corresponding computation." Dijkstra letter to CACM.

ALGOL 60

- GO TO 99
- IF ... THEN ... ELSE .... (hierarchical)
- for i := 3, 7, 11 step 1 until 16, i/2 while i >= 1, 2 step i until 32 do ..
  - BAROQUE, all expressions re-eval each time through loop:
  - 3, 7, 11, 12, 13, 14, 15, 16, 8, 4, 2, 1, 2, 4, 8, 16, 32.
- switch - like in C/C++/Java.

Pascal

- go to
- if .. then .. else
- for, while, repeat (confusion w/positive vs. negative exit)
- labelled case - Tony Hoare
  - clear & efficient
  - construct jump table,
  - optimize depending on size,
  - self-documenting.

More on Case

- Modula 2 improved by adding otherwise clause
- ML's pattern matching is compiled into a case statement:

```
fun reverse l = case l of
        nil => nil |
        (h::rest) => (reverse rest)@[h];
```

- if-then-else as well

Ada

```
iteration specification loop
   body
end loop.
```

- where iteration specification can be:
  - while condition,
  - for vbl in discrete range
    e.g. for i in 1..10 loop .. end loop
- exit and “exit when”