#### Lecture #38

**Next Friday:** HKN surveys. Extra points awarded to those who participate!

**Today:** A little side excursion into nitty-gritty stuff: Storage management.

Last modified: Wed Nov 26 12:56:27 2014

CS61B: Lecture #38 1

## Explicit vs. Automatic Freeing

- Java has no means to free dynamic storage.
- However, when no expression in any thread can possibly be influenced by or change an object, it might as well not exist:

```
IntList wasteful ()
{
   IntList c = new IntList (3, new IntList (4, null));
   return c.tail;
   // variable c now deallocated, so no way
   // to get to first cell of list
}
```

• At this point, Java runtime, like Scheme's, recycles the object c pointed to: garbage collection.

## Scope and Lifetime

- Scope of a declaration is portion of program text to which it applies (is visible).
  - Need not be contiguous.
  - In Java, is static: independent of data.
- Lifetime or extent of storage is portion of program execution during which it exists.
  - Always contiguous
  - Generally dynamic: depends on data
- Classes of extent:
  - Static: entire duration of program
  - Local or automatic: duration of call or block execution (local variable)
  - Dynamic: From time of allocation statement (new) to deallocation, if any.

Last modified: Wed Nov 26 12:56:27 2014

CS61B: Lecture #38 2

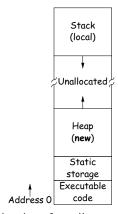
#### Under the Hood: Allocation

- Java pointers (references) are represented as integer addresses.
- Corresponds to machine's own practice.
- ullet In Java, cannot convert integers  $\leftrightarrow$  pointers,
- But crucial parts of Java runtime implemented in C, or sometimes machine code, where you can.
- Crude allocator in C:

```
char store[STORAGE_SIZE]; // Allocated array
size_t remainder = STORAGE_SIZE;

/** A pointer to a block of at least N bytes of storage */
void* simpleAlloc (size_t n) { // void*: pointer to anything
  if (n > remainder) ERROR ();
  remainder = (remainder - n) & ~0x7; // Make multiple of 8
  return (void*) (store + remainder);
}
```

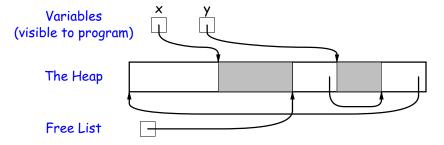
# Example of Storage Layout: Unix



- OS gives way to turn chunks of unallocated region into heap.
- Happens automatically for stack.

#### Free Lists

- Explicit allocator grabs chunks of storage from OS and gives to applications.
- Or gives recycled storage, when available.
- When storage is freed, added to a *free list* data structure to be recycled.
- Used both for explicit freeing and some kinds of automatic garbage collection.
- Problem: free memory fragments.



## **Explicit Deallocating**

- C/C++ normally require explicit deallocation, because of
  - Lack of run-time information about what is array
  - Possibility of converting pointers to integers.
  - Lack of run-time information about unions:

```
union Various {
  int Int;
  char* Pntr;
  double Double;
} X; // X is either an int, char*, or double
```

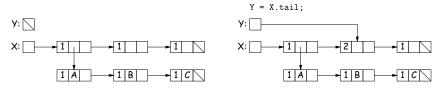
- Java avoids all three problems; automatic collection possible.
- Explicit freeing can be somewhat faster, but rather error-prone:
  - Memory corruption
  - Memory leaks

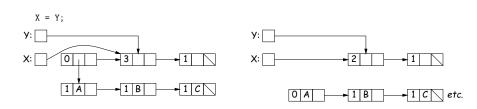
Last modified: Wed Nov 26 12:56:27 2014

# Garbage Collection: Reference Counting

CS61B: Lecture #38 6

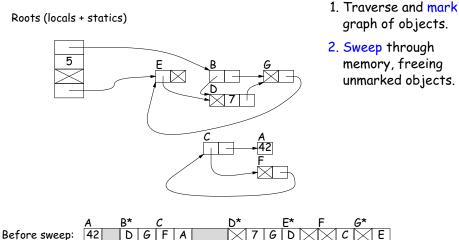
• Idea: Keep count of number of pointers to each object. Release when count goes to 0.





Last modified: Wed Nov 26 12:56:27 2014 C561B: Lecture #38 7 Last modified: Wed Nov 26 12:56:27 2014 C561B: Lecture #38 8

### Garbage Collection: Mark and Sweep



Last modified: Wed Nov 26 12:56:27 2014

After sweep:

DG

CS61B: Lecture #38 9

### Copying Garbage Collection

- Mark-and-sweep algorithms don't move any exisiting objects—pointers stay the same.
- The total amount of work depends on the amount of memory swept—
   i.e., the total amount of active (non-garbage) storage + amount of
   garbage. Not necessarily a big hit: the garbage had to be active at
   one time, and hence there was always some "good" processing in the
   past for each byte of garbage scanned.
- Another approach: copying garbage collection takes time proportional to amount of active storage:
  - Traverse the graph of active objects breadth first, copying them into a large contiguous area (called "to-space").
  - As you copy each object, mark it and put a forwarding pointer into it that points to where you copied it.
  - The next time you have to copy a marked object, just use its forwarding pointer instead.
  - When done, the space you copied from ("from-space") becomes the next to-space; in effect, all its objects are freed in constant time.

# Copying Garbage Collection Illustrated

