

Scope and Lifetime

Declaration is portion of program text to which it applies

Can be contiguous.

Can be static: independent of data.

extent of storage is portion of program execution during which it exists.

Can be contiguous

Can be dynamic: depends on data

Can be static:

Can be lifetime duration of program

automatic: duration of call or block execution (local variables)

From time of allocation statement (**new**) to deallocation.

1:52:05 2016

CS61B: Lecture #39 2

Under the Hood: Allocation

References (references) are represented as integer addresses.

Due to machine's own practice.

Do not convert integers ↔ pointers,

Parts of Java runtime implemented in C, or sometimes C++, where you can.

For in C:

```
[STORAGE_SIZE]; // Allocated array
pointer = STORAGE_SIZE;

// pointer to a block of at least N bytes of storage */
void* malloc(size_t n) { // void*: pointer to anything
    if (n < 0) return (remainder) ERROR();
    int remainder = (n - 1) & ~0x7; // Make multiple of 8
    return (void*) (store + remainder);
}
```

1:52:05 2016

CS61B: Lecture #39 4

Explicit Deallocating

Programs may require explicit deallocation, because of

lack of in-time information about what is array

lack of converting pointers to integers.

lack of in-time information about *unions*:

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

lack of all three problems; automatic collection possible.

Explicit deallocation can be somewhat faster, but rather error-prone:

Memory corruption

Memory leaks

1:52:05 2016

CS61B: Lecture #39 6

Lecture #39

Surveys. [Extra points awarded to those who participate!](#)

Side excursion into nitty-gritty stuff: Storage management

1:52:05 2016

CS61B: Lecture #39 1

Explicit vs. Automatic Freeing

Explicit means to free dynamic storage.

When no expression in any thread can possibly be influenced by an object, it might as well not exist:

```
def freeList() {
```

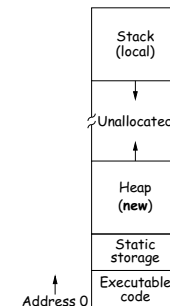
```
    c = new IntList(3, new IntList(4, null));
    tail = c;
    // Note: c now deallocated, so no way to reach
    // tail to first cell of list
}
```

But, Java runtime, like Scheme's, recycles the object c via garbage collection.

1:52:05 2016

CS61B: Lecture #39 3

Example of Storage Layout: Unix



Programs may need to turn chunks of unallocated region into heap.

Programs may need to free memory automatically for stack.

1:52:05 2016

CS61B: Lecture #39 5

Free List Strategies

requests generally come in multiple sizes.

requests on the free list are big enough, and one may have to split a chunk and break it up if too big.

strategies to find a chunk that fits have been used:

Best fit:

checks in LIFO or FIFO order, or sorted by address.

combines adjacent blocks.

works for **first fit** on list, **best fit** on list, or **next fit** on list to choose the best-chosen chunk.

Worst fit: separate free lists for different chunk sizes.

Sticky items: A kind of segregated fit where some newly added blocks of one size are easily detected and combined with other chunks.

using blocks reduces **fragmentation** of memory into lots of little free chunks.

Free Lists

allocator grabs chunks of storage from OS and gives to

program. Cycled storage, when available.

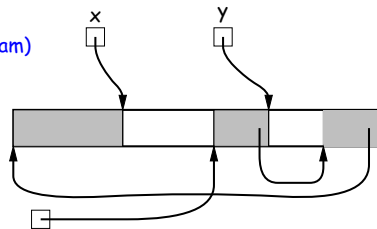
When memory is freed, added to a *free list* data structure to be

used for explicit freeing and some kinds of automatic garbage

variables (pointers to program)

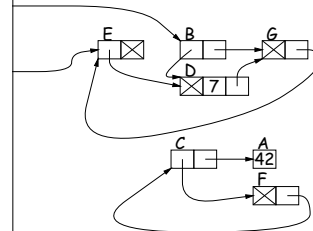
on the Heap

Free List

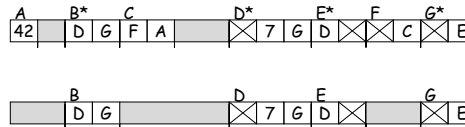


Garbage Collection: Mark and Sweep

(techniques)



1. Traverse and **mark** the graph of objects.
2. **Sweep** through memory, freeing unmarked objects.



Copying Garbage Collection

Approach: **copying garbage collection** takes time proportional to amount of active storage:

Traverse the graph of active objects breadth first, copying them to a contiguous area (called "to-space").

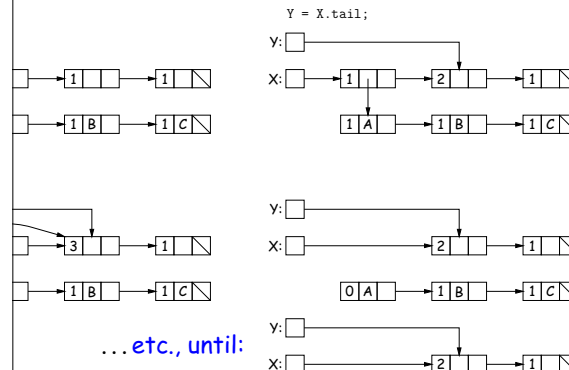
Copy each object, mark it and put a *forwarding pointer* that points to where you copied it.

Next time you have to copy an already marked object, just put a forwarding pointer instead.

Eventually, the space you copied from ("from-space") becomes to-space; in effect, all its objects are freed in constant

Garbage Collection: Reference Counting

Keep a count of number of pointers to each object. Release memory when count goes to 0.



Cost of Mark-and-Sweep

Reference counting algorithms don't move any existing objects—pointers are updated.

The amount of work depends on the amount of memory swept—proportional to amount of active (non-garbage) storage + amount of garbage. It's not necessarily a big hit: the garbage had to be active at some point, hence there was always some "good" processing in the heap for every byte of garbage scanned.

Objects Die Young: Generational Collection

Objects stay active, and need not be collected.

Need to avoid copying them over and over.

Generational garbage collection schemes have two (or more) spaces: one for newly created objects (*new space*) and one for objects that have survived garbage collection (*old space*).

Generational garbage collection collects only in new space, ignores pointers that point to old space, and moves objects to old space.

Uses usual roots plus pointers in old space that have changed (or might be pointing to new space).

When new space full, collect all spaces.

This leads to much smaller *pause times* in interactive systems.

Copying Garbage Collection Illustrated

from:

A	B	C	D	E	F	G
42	D	G	F	A	7	G

 B: Old object
B': New object
*: marked

to:

--	--	--	--	--	--	--

forwarding pointers

from:

A	B*	C	D	E*	F	G
42	B'	G	F	A	7	G

 Copy roots

to:

B'	E'					
D	G	D				

from:

A	B*	C	D*	E*	F	G*
42	B'	G	F	A	7	G

 Copy from to-space in (b). Only D is new

to:

B'	E'	D'				
D	G	D	7	G	E	

from:

A	B*	C	D*	E*	F	G*
42	B'	G	F	A	7	G

 Copy from to-space in (c). No new objects

There's Much More

Next highlights.

Focus on how to implement these ideas efficiently.

Generational garbage collection: What if objects scattered over many spaces?

Incremental garbage collection: where predictable pause times are important, doing a little at a time.