Concepts of Programming Languages

Lecture 17 - Memory Management

Patrick Donnelly

Montana State University

Spring 2014

Administrivia

Assignments:

Programming #3 : due 04.14

Homework #4 : due 04.16

Reading:

Chapter 6.11

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.

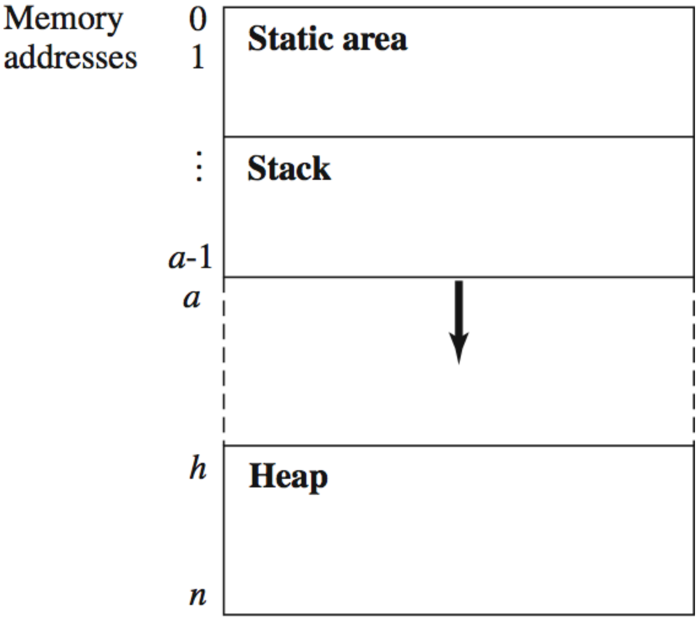
B. Stroustrup

The Heap

The major areas of memory:

- **Static area**: fixed size, fixed content, allocated at compile time
- **Run-time stack**: variable size, variable content, center of control for function call and return
- **Heap**: fixed size, variable content, dynamically allocated objects and data structures

Structure of Run-Time Memory



Allocating Heap Blocks

The function `new` allocates a block of heap space to the program.

Example

`new(5)` returns the address of the next block of 5 words available in the heap:

h

7	*undef*	12	0
3	*unused*	*unused*	*unused*
undef	0	*unused*	*unused*
unused	*unused*	*unused*	*unused*

...

n

h

7	*undef*	12	0
3	*unused*	*unused*	*unused*
undef	0	*undef*	*undef*
undef	*undef*	*undef*	*unused*

...

n

Stack and Heap Overflow

Definition

Stack overflow occurs when the top of stack, `a`, would exceed its (fixed) limit, `h`.

Definition

Heap overflow occurs when a call to `new` occurs and the heap does not have a large enough block available to satisfy the call.

Implementation of Dynamic Arrays

Consider the declaration `int A[n];`

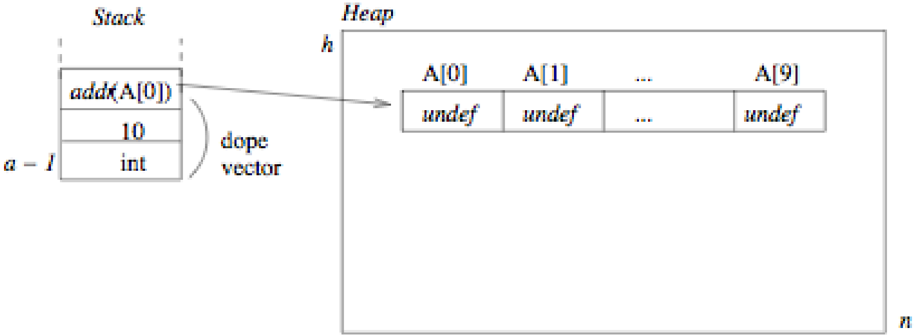
Its meaning (Meaning Rule 11.1) is:

1. Compute `addr(A[0]) = new(n)`.
2. Push `addr(A[0])` onto the stack.
3. Push `n` onto the stack.
4. Push `int` onto the stack.

Step 1 creates a heap block for `A`.

Steps 2-4 create the dope vector for `A` in the stack.

Stack and Heap Allocation for `int A[10]`



Array References

Meaning Rule 11.2 The meaning of an ArrayRef `ar` for an array declaration `ad` is:

1. Compute `addr(ad[ar.index]) = addr(ad[0])+ar.index-1`
2. If `addr(ad[0])`≤`addr(ad[ar.index])<addr(ad[0])+ad.size`, return the value at `addr(ad[ar.index])`
3. Otherwise, signal an index-out-of-range error.

Array References

Meaning Rule 11.2 The meaning of an ArrayRef `ar` for an array declaration `ad` is:

1. Compute `addr(ad[ar.index]) = addr(ad[0])+ar.index-1`
2. If `addr(ad[0])≤addr(ad[ar.index])<addr(ad[0])+ad.size`, return the value at `addr(ad[ar.index])`
3. Otherwise, signal an index-out-of-range error.

Example

Consider the `ArrayRef A[5]`. The value of `A[5]` is addressed by `addr(A[0])+4`.

Note: this definition includes run-time range checking.

Array Assignments

Meaning Rule 11.3 The meaning of an Assignment as is:

1. Compute `addr(ad[ar.index])=addr(ad[0])+ar.index-1`
2. If `addr(ad[0])`$\leq$`addr(ad[ar.index])<addr(ad[0])+ad.size` then assign the value of `as.source` to `addr(ad[ar.index])`.
3. Otherwise, signal an index-out-of-range error.

Array Assignments

Meaning Rule 11.3 The meaning of an Assignment as is:

1. Compute `addr(ad[ar.index])=addr(ad[0])+ar.index-1`
2. If `addr(ad[0])`$\leq$`addr(ad[ar.index])<addr(ad[0])+ad.size` then assign the value of `as.source` to `addr(ad[ar.index])`.
3. Otherwise, signal an index-out-of-range error.

Example

The assignment `A[5]=3` changes the value at heap address `addr(A[0])+4` to 3, since

`ar.index=5 and addr(A[5])=addr(A[0])+4.`

Garbage Collection

Definition

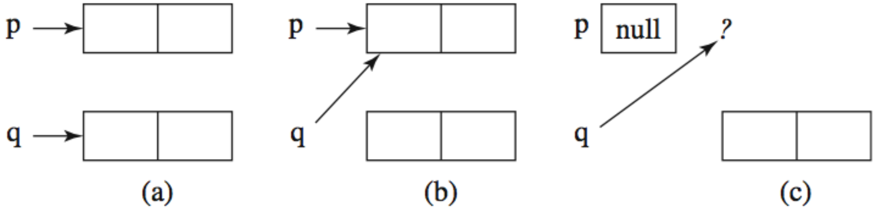
Garbage is a block of heap memory that cannot be accessed by the program.

Garbage can occur when either:

1. An allocated block of heap memory has no reference to it (an “orphan”), or
2. A reference exists to a block of memory that is no longer allocated (a “widow”).

Garbage Example

```
class node {
     int value;
     node next;
}
node p, q;
p = new node();
q = new node();
q = p;
delete p;
```



Garbage Collection Algorithms

Definition

Garbage collection is any strategy that reclaims unused heap blocks for later use by the program.

Three classical garbage collection strategies:

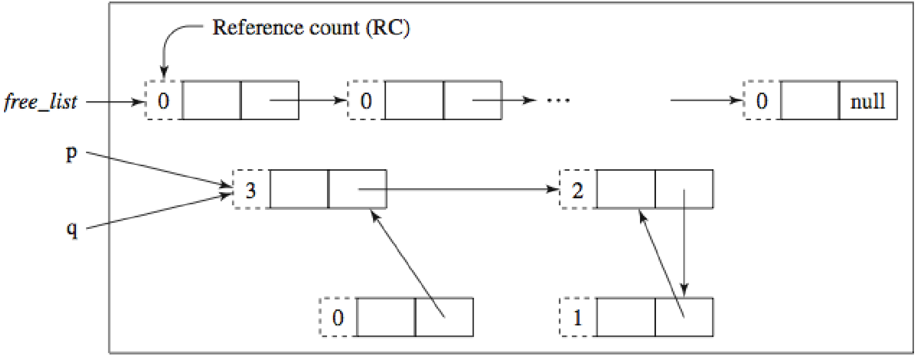
- **Reference Counting** - occurs whenever a heap block is allocated, but doesn't detect all garbage.
- **Mark-Sweep** - Occurs only on heap overflow, detects all garbage, but makes two passes on the heap.
- **Copy Collection** - Faster than mark-sweep, but reduces the size of the heap space.

Reference Counting

The heap is a chain of nodes (the *free_list*).

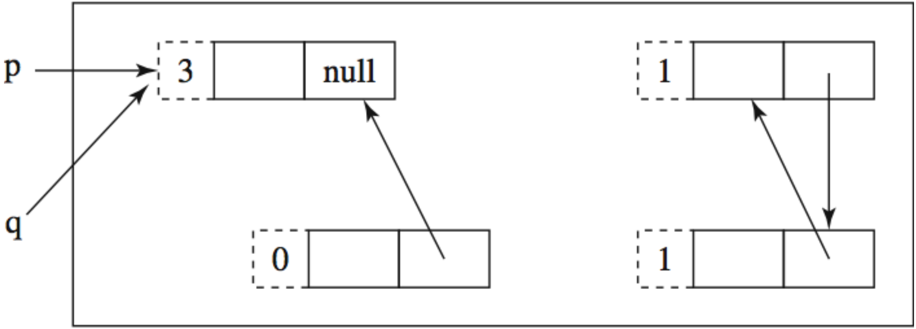
Each node has a reference count (RC).

For an assignment, like `q = p`, garbage can occur:



But not all garbage is collected. . .

Since q's node has RC=0, the RC for each of its descendants is reduced by 1, it is returned to the *free_list*, and this process repeats for its descendants, leaving:



Note the orphan chain on the right.

Mark-Sweep

Each node in the *free_list* has a mark bit (MB) initially 0.

Called only when heap overflow occurs:

Pass I: Mark all nodes that are (directly or indirectly) accessible from the stack by setting their MB=1.

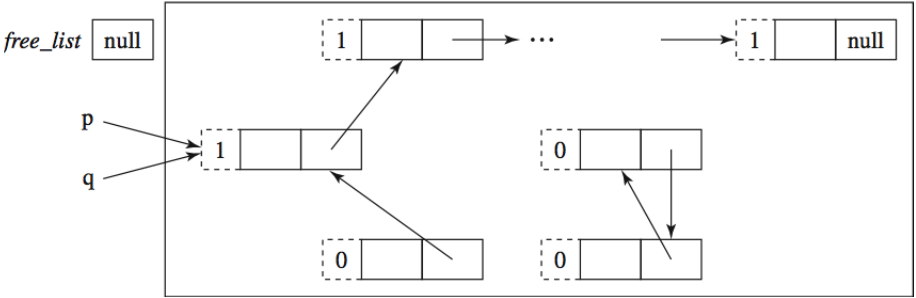
Pass II: Sweep through the entire heap and return all unmarked (MB=0) nodes to the free list.

Note: all orphans are detected and returned to the free list.

Heap after Pass I of Mark-Sweep

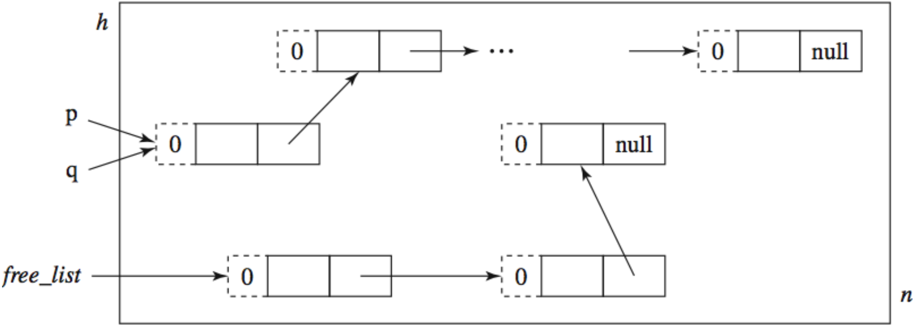
Triggered by `q=new node()` and *free_list* = null.

All accessible nodes are marked 1.



Heap after Pass II of Mark-Sweep

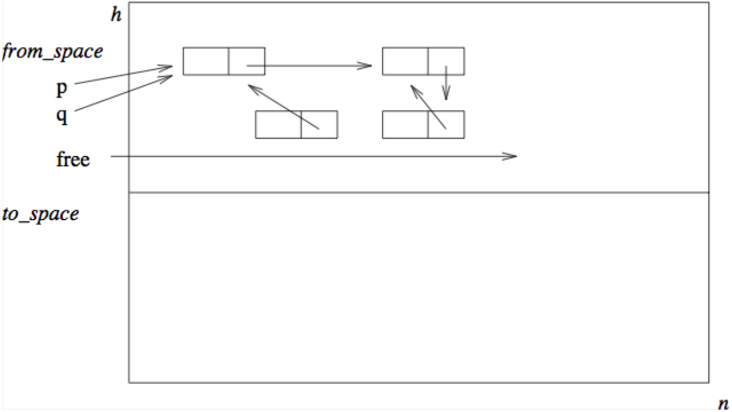
Now *free_list* is restored and the assignment `q=new node()` can proceed.



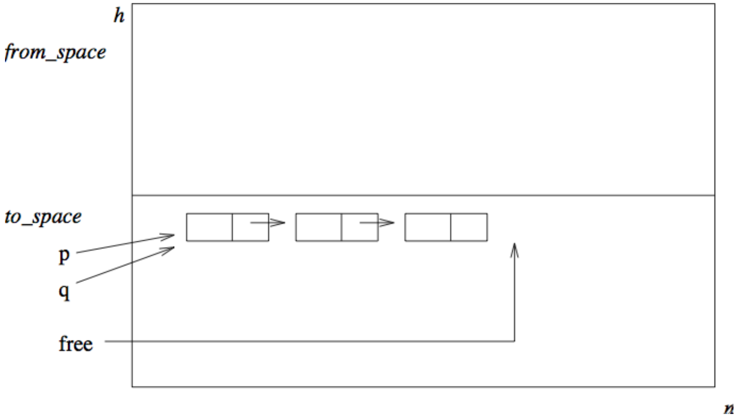
Copy Collection

Heap partitioned into two halves; only one is active.

Triggered by `q=new node()` and *free_list* outside the active half:



Accessible nodes copied to other half



Note: The accessible nodes are packed, orphans are returned to the free_list, and the two halves reverse roles.

Garbage Collection Summary

Modern algorithms are more elaborate.

- Most are hybrids/refinements of the above three.

In Java, garbage collection is built-in.

- runs as a low-priority thread.
- Also, `System.gc` may be called by the program.

Functional languages have garbage collection built-in.

C/C++ default garbage collection to the programmer.