Heap Heap Hooray: Memory Management

Tyler Gutowski, Trevor Schiff,
Dr. Ryan Stansifer (client)

Task		Tyler	Trevor
Heap	Added HeapHeader to save information about the Objects for the GC.	0.0	1.0
Allocator	Added methods for heap allocation management (allocation, freeing, checking addresses), as well as increment and decrement RC.	0.2	0.8
Reference Counting Initialization Handling	Added functionality to ensure an object is initialized with a counter.	0.5	0.5
Reference Counting Reference Handling	RC gets incremented when a reference is passed.	0.5	0.5
Reference Counting Argument Handling	RC gets incremented when an object is passed in as a function argument.	0.5	0.5
Environment Setup (SPARC, QEMU, Jabberwocky)	Setting up SPARC on a local machine to run MiniJava compiler. We chose local over Andrew because we might need specific permissions at a later date.	1.0	0.0
Memory Tests	Run MiniJava tests to see when the reference counter increments and decrements.	0.5	0.5
Mark-and-Sweep	Secondary task	0.0	0.0

Basic Overview: Reference Counting

Objects are initialized with a reference count of 0

If a reference to an object is created

(Object initialized, referenced, or passed as a function param)

- The object’s reference count will increment

If a reference to an object is destroyed

(Object leaves scope or is manually freed)

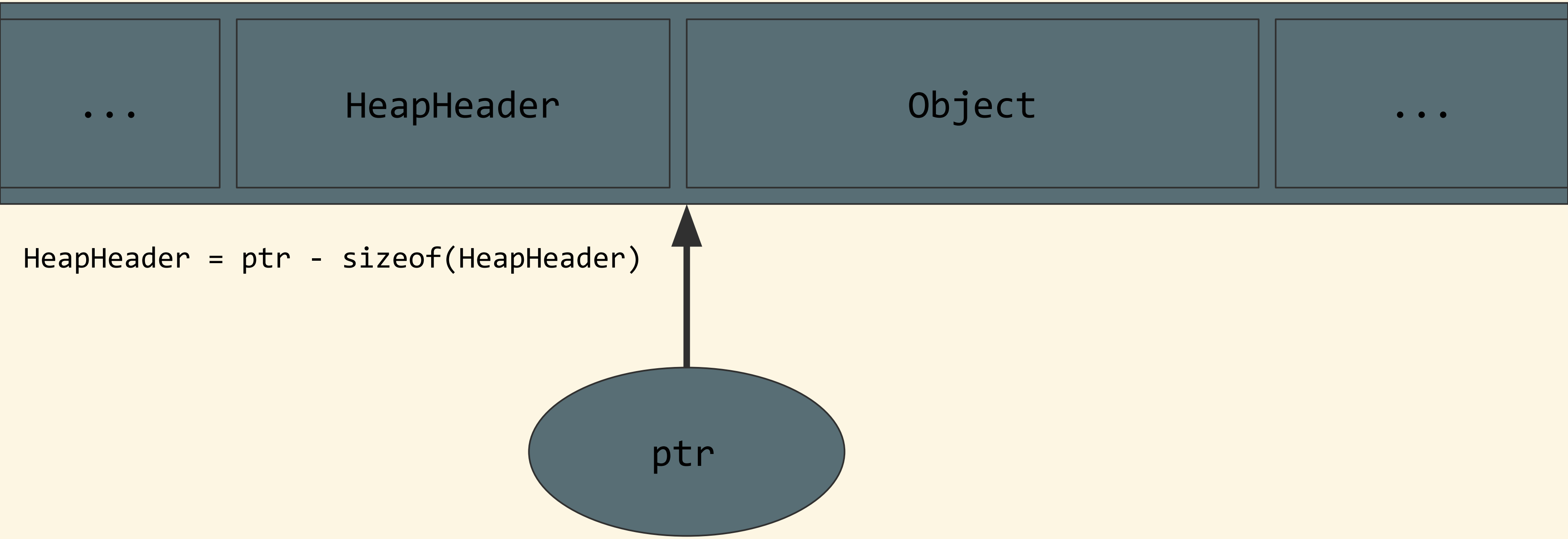
- The object’s reference count will decrement
  - If the object’s reference count is now zero
    - Memory allocated for the object is now freed

Implementation: Reference Counting

```
typedef struct HeapHeader {
    u32 tag; // at 0x0, 4 bytes (Block identification)
    struct HeapHeader* next; // at 0x4, 4 bytes
    struct HeapHeader* prev; // at 0x8, 4 bytes
    u32 size; // at 0xC, 4 bytes (Size of this allocation)
    s32 marked : 1; // at 0x10, 1 bit (Mark bit (For mark-sweep)
    volatile s32 ref : 31; // at 0x10, 31 bits (Reference count)
    u8 data[]; // (Block data)
} HeapHeader;
```

Size of 20 bytes total

Implementation: Reference Counting



HeapHeader = ptr - sizeof(HeapHeader)

Milestone 3 Goals

- Mark-sweep GC as primary implementation
    - RC cannot collect cyclic (self-referencing) garbage
    - Traverses heap block graph by searching block data for pointers
- Copying GC as secondary implementation
    - Mark-sweep fragments the memory, which leaves some of it unusable
    - Copying defragments
- Run tests with all 3 GC implementations
    - Compare tests with vital metrics