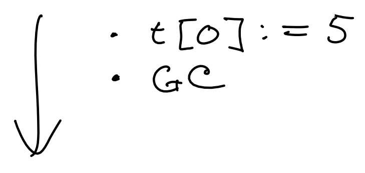
Memory Management

1. User allocates & deallocates
2. We do

* What invariants do we have?
* Whose job is it to maintain them?

Garbage collection: mark/compact

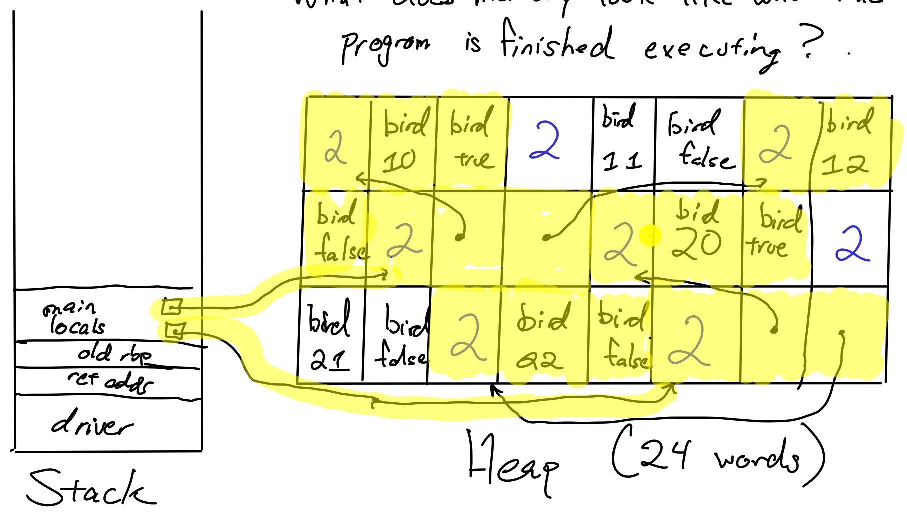
Falcon



Gull

```
def f n =
  let a = (n, true) in
  let b = (n+1, false) in
  let c = (n+2, false) in
  (a, c)
end

(f 10, f 20)
```



What does memory look like when this program is finished executing?

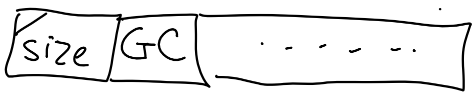
(((10, true), (12, false)),
 ((20, true), (22, false)))

"Liveness"

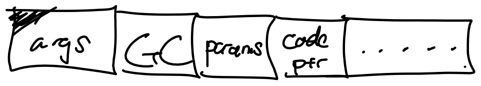
heap_cursor: everything left is allocated; everything right is not

no fragmentation, simple invariant

Tuples



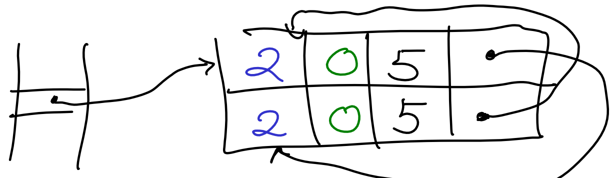
Closures

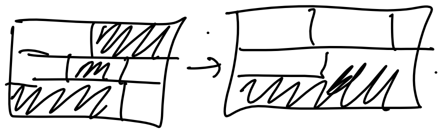


Invariant: during normal run of program, GC word = 0

Strategy: when I need memory and I don't have enough, run GC alg. to find free memory and shift everything to the left

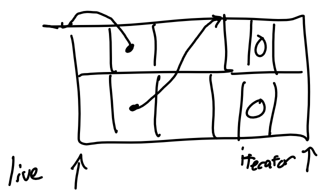
Phase 1: Mark
Phase 2: Forward
Phase 3: Update
Phase 4: Compact
Phase 5: Unmark



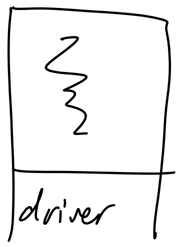


Phase 1:
From each pointer on the stack, DFS and mark each heap object you find. (GC word = 1)

Phase 2:
Iterate through heap and set GC word for each object to mem location where I want it to be.
If heap object is not alive, $\underset{\text{word}}{\text{GC}}$ = 0.

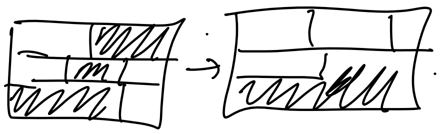


Phase 3: (Update ptrs)
Iterate through stack and heap looking for $\overset{\text{bird}}{\text{ptrs}}$ into heap. Replace their values with the value in the GC word of the object they point to.



Phase 4: Compact heap

Do this now:



Phase 5: Unmark

Set every GC word to 0