```
| EPair(e1, e2) ->
  let e1_is = e_to_is e1 si env false in
  let e2_is = e_to_is e2 (si + 1) env false in
  let save_e1 = sprintf "mov [rsp-%d], rax" (stackloc si) in
  let save_e2 = sprintf "mov [rsp-%d], rax" (stackloc (si + 1)) in
  e1_is @ [save_e1] @ e2_is @ [save_e2] @ [
    sprintf "mov rax, [rsp-%d]" (stackloc si);
    sprintf "mov [r15], rax";
    sprintf "mov rax, [rsp-%d]" (stackloc (si + 1));
    sprintf "mov [r15 + 8], rax";
    sprintf "mov rax, r15";
    sprintf "add r15, 16";
  ]
| EFst(e) ->
  let e_is = e_to_is e si env false in
  e_is @ [sprintf "mov rax, [rax]"]
| ESnd(e) ->
  let e_is = e_to_is e si env false in
  e_is @ [sprintf "mov rax, [rax+8]"]
| ESetFst(e_pair, e_val) ->
  let e1_is = e_to_is e_pair (si + 1) env false in
  let e2_is = e_to_is e_val si env false in
  let save_e1 = sprintf "mov [rsp-%d], rax" (stackloc si) in
  e1_is @ [save_e1] @ e2_is @ [
    sprintf "mov rbx, [rsp-%d]" (stackloc si);
    sprintf "mov [rbx], rax"]
| ESetSnd(e_pair, e_val) ->
  let e1_is = e_to_is e_pair (si + 1) env false in
  let e2_is = e_to_is e_val si env false in
  let save_e1 = sprintf "mov [rsp-%d], rax" (stackloc si) in
  e1_is @ [save_e1] @ e2_is @ [
    sprintf "mov rbx, [rsp-%d]" (stackloc si);
    sprintf "mov [rbx+8], rax"]
```

no malloc

update heap-alloc'd value

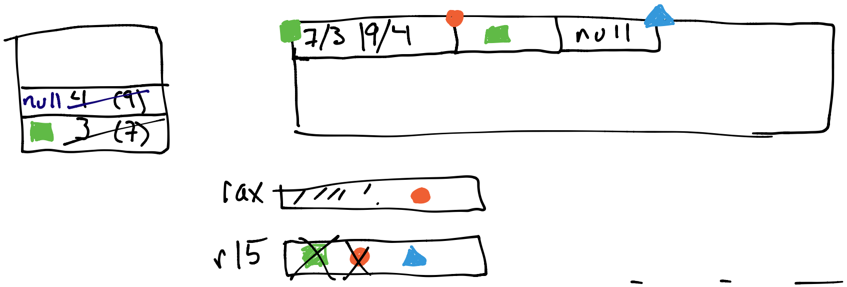
```
int64_t read_num() {
  // Read and return a number from the user
}

void print(int64_t val) {
  if((val & 1)) {
    printf("%lld", (val - 1) / 2);
  } else if(val == 6) {
    printf("true");
  } else if(val == 2) {
    printf("false");
  } else if(val == 0) {
    printf("null");
  } else if((val & 7) == 0) {
    int64_t* as_ref = (int64_t*)val;
    printf("(pair ");
    print(as_ref[0]);
    printf(" ");
    print(as_ref[1]);
    printf(")");
  } else {
    printf("Weird value: %lld", val);
  }
}

int main(int argc, char** argv) {
  int64_t* HEAP = calloc(sizeof(int64_t), 1000);
  int64_t result = our_code_starts_here(HEAP);
  print(result);
  printf("\n");
  return 0;
}
```

(read_line)

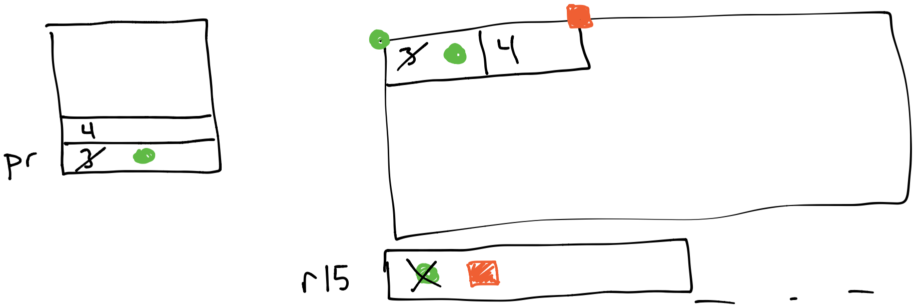
null is 0

For each of the following programs, what will the stack and heap look like just before the final ret?

```
(pair (pair 3 4) null)
```



```
(let (pr (pair 3 4))
  (set-first pr pr))
```



```
(def range (n : Num m : Num)
  (if (< m n)
    null
    (pair n (range (+ n 1) m))))
(let (r (range 4 7)) → print here?
  (set r (range 6 8))
  r))
```

(range 4 7)

→

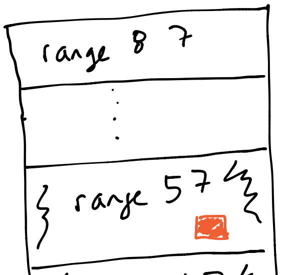
(pair 4 (range (+ 4 1) 7))

→

(pair 4 (pair 5 (range (+ 5 1) 7)))

...

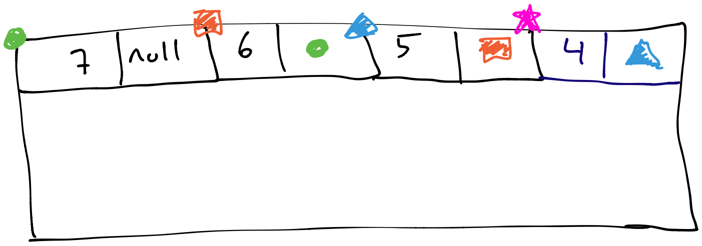
(pair 4 (pair 5 (pair 6 (pair 7 null))))



Will 6 appear before or after 4 on the heap?

A: Lower addr for 6

B: Higher addr for 6



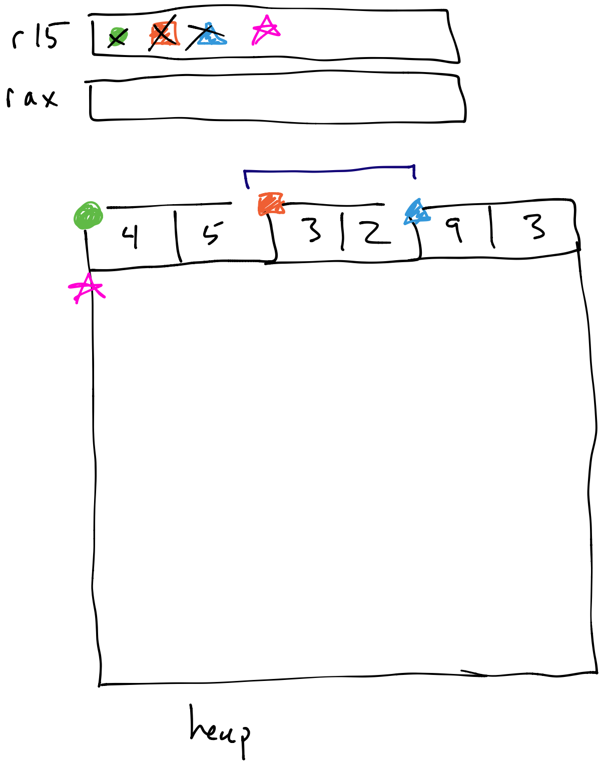
In this program, what does the stack and heap look like when n = 1 and we update it to 0 with the set in the while loop?

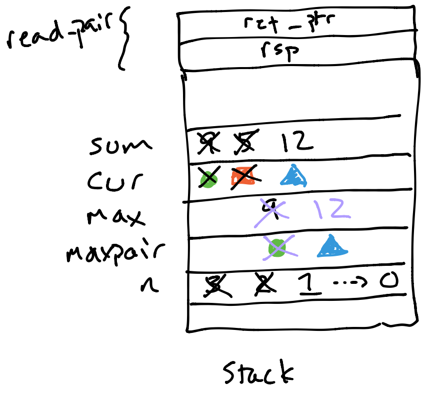
```
(def read_pair () : (Pair Num Num)
  (pair (read_num) (read_num)))
(def maxof (n : Num) : (Pair Num Num)
  (let ((max 0) (maxpair null))
    (while (> n 0)
      (let ((cur (read_pair)) ; reads a pair from user input
            (sum (+ (first cur) (second cur))))
    --> (set n (- n 1))
        (if (> sum max)
            (let () (set max sum) (set maxpair cur))
            null))))
    maxpair))
(maxof input)
```

(free maxpair)

(free cur)

```
$ ./maxof.run 3
4 5
3 2
9 3
(pair 9 3)
```



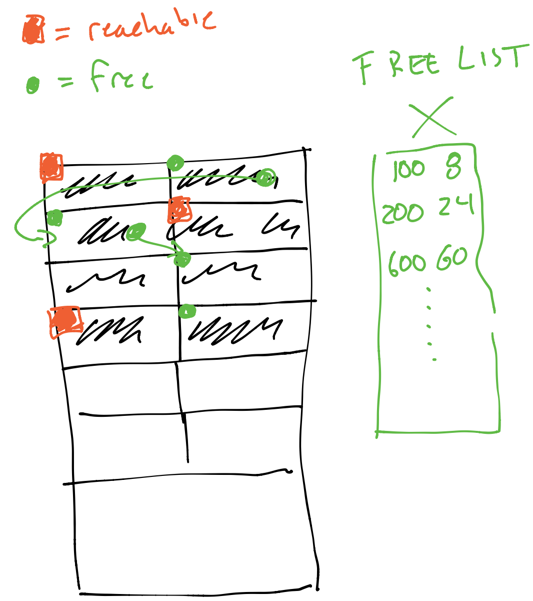


$ ./maxof.run 10000

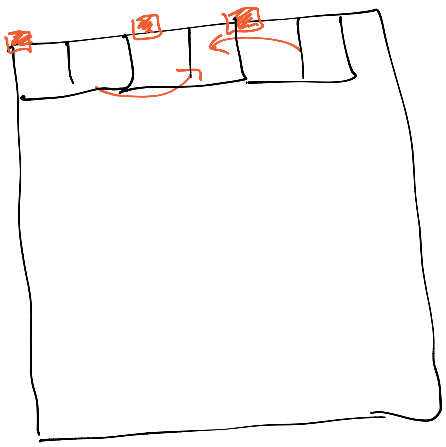
if there are no references to a heap loc REACHABLE from the stack, free it

Every once in a while, stop program, find all reachable references, and "clean up" everything else.

Garbage Collection / Automated MM



mark/compact



move reachable stuff to beg.