1 Garbage Collection

Now that we're working with memory in the heap, let's suppose we *don't* have a lot of memory to work with. In this case, we need to think about *garbage collection* as a way to get rid of unused memory so we can allocate memory for useful things.

1.1 Motivation

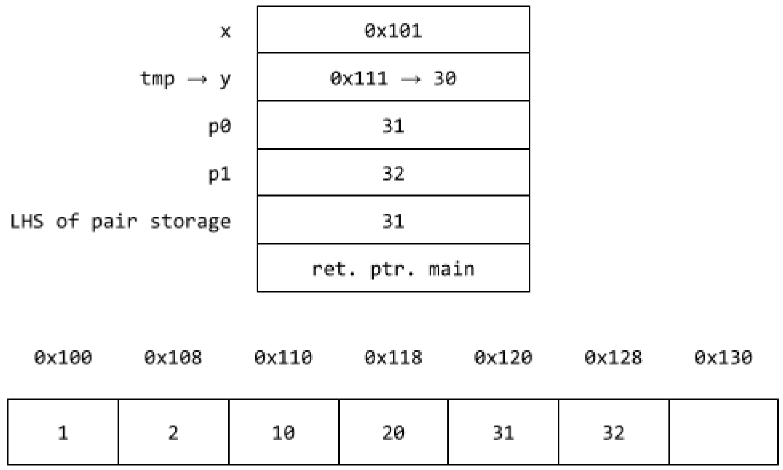
Let's take a look at two examples to get an idea of what we're working with.

1.1.1 Motivation 1: Basic Garbage

Recall the following code from class:

```
(let (x (pair 1 2))
    (let (y (let (tmp (pair 10 20)) (+ (fst tmp) (snd tmp))))
        (let (p0 (+ (fst x) y))
            (let (p1 (+ (snd x) y))
                (pair p0 p1)
            )
        )
    )
)
```

After creating the final pair, the memory diagram looks like[1][2]



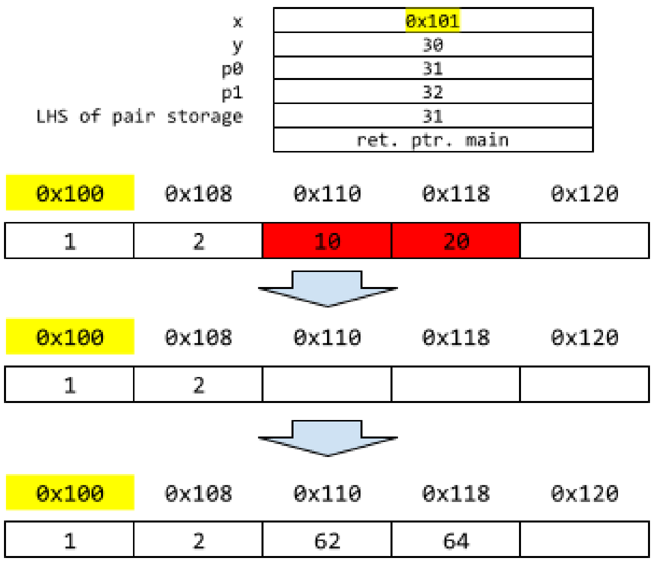
Here, the stack is the top diagram while the heap is the bottom diagram. Note that `rax`[3] is `0x121`, and `r15`[4] is `0x130`.

Now, let's suppose our heap only has five available words. `rax` will hold the result of `(+ (snd x) y)` (i.e., result of addition, which is a number). Immediately, we should notice that

- We don't have enough memory to allocate for the final pair!
- More importantly, however, the values in the heap at location `0x110` and `0x118` are **garbage**. Nothing in the stack (or a register) is referring to these values!

[1]For the sake of conciseness, we're showing the numbers without their tagged representation.
[2]Also, the `tmp` to `y` arrow indicates that we're *reusing* that space.
[3]Which is storing the answer to our program
[4]Recall that `r15` will always point to the next available word in the heap

To clarify, the idea is that any memory that is not reachable from the stack (or any registers) is considered garbage and can be reused. So, our goal is to get rid of the garbage so we have enough memory to allocate for the final pair. With this said, a high-level implementation of a basic garbage collector would look something like this.



So here's what's going on:

- We've determined that the stuff at `0x110` and `0x118` are garbage, so we can get rid of them.
- After that, we can *compact the heap*, essentially moving `r15` back to `0x110`. After that, we can allocate memory for our next pair.
- This gives us the desired result, with `rax` being `0x110`.

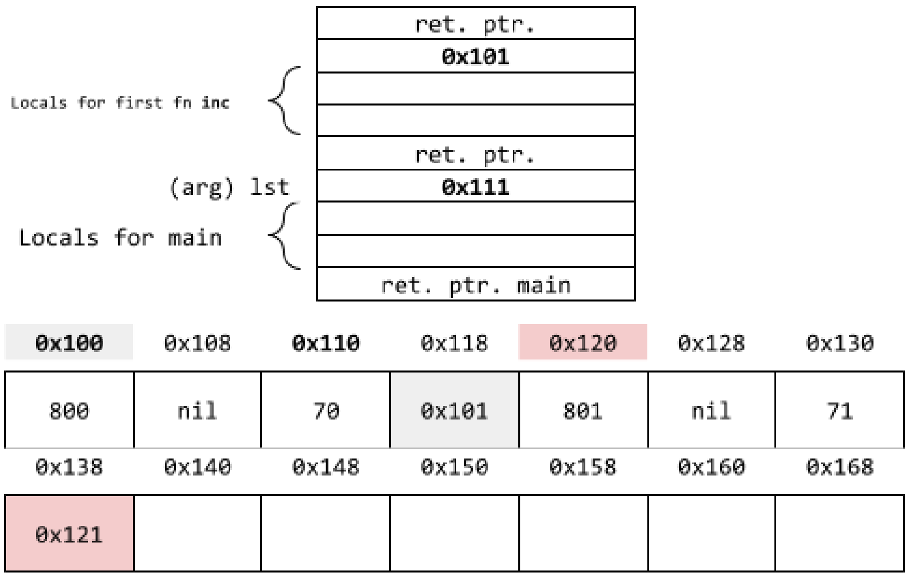
A key observation here is that we didn't need to "fix" any memory addresses stored in the stack or in the heap itself.

1.1.2 Motivation 2: Slightly Complicated Garbage

Consider the following code:

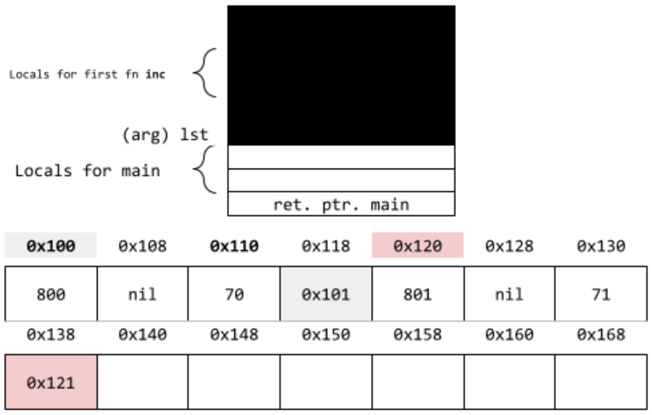
```
(fun (inc lst)
    (if (= lst nil)
        nil
        (pair (+ (fst lst) 1) (inc (snd lst)))
    )
)

(inc (inc (pair 70 (pair 800 nil))))
```

After the second call to `inc` (i.e., after the first call to `inc` finishes), we have the following rough diagram:



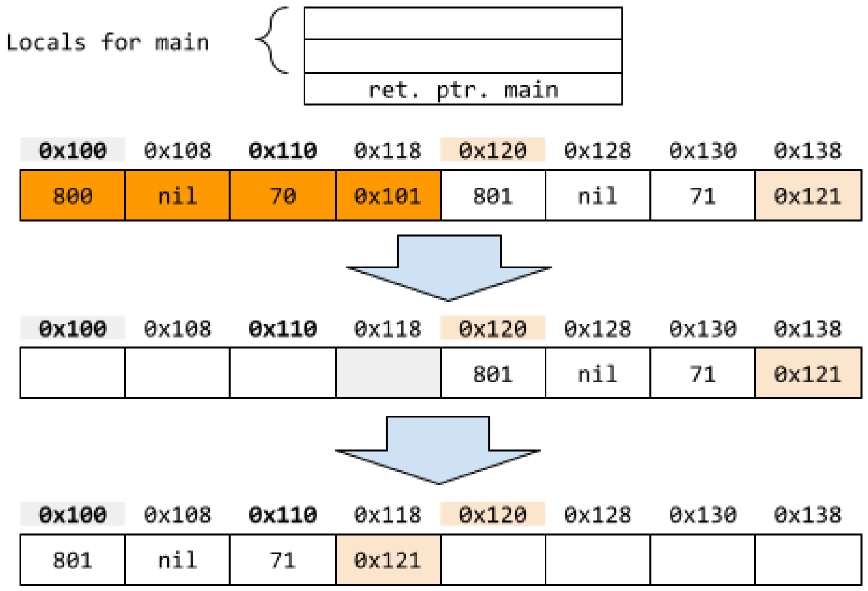
The register `rax` would have value `0x131`. What is considered garbage? The first four words – `0x100` through `0x118` – are considered **garbage** since there's no references to those words anywhere in the stack. However, why is this the case?

- We'll define the stack as everything at an address higher than `rsp` to the top of `our_code_starts_here`, but not anything lower address or above that. Therefore, we don't need to consider the following in the stack when deciding what is garbage:



- So, as long as no references to `0x111` is in main, we can prove that it's garbage.

Now, let's suppose our heap only has eight available words. In this example, `rax` is `0x131`. As one might have suspected, we don't have any memory left to allocate the remaining pairs needed for this program; in particular, after we call the `inc` function with the pair that we got from our initial call to `inc`, we don't have enough memory to allocate another pair needed for the recursive call. So, we need to collect some garbage. Here's how we might go about this.



So, here's what's going on:

- We determined that the first four words are garbage, since no items in the stack or any registers are referring to those four words in the heap.
- We can compact the heap by moving `r15` to the beginning of our heap, thus allowing us to reuse the four words that are garbage.
- Now that we have room in the heap, we can allocate the final pair and change `rax` to point to our final result.

Well, *not quite*. Unlike the previous example, notice how we have a memory address in the heap that's referring to a memory address that's now garbage. That memory address has been relocated, so **we need to fix this.** Specifically, we need to change the value at `0x118` to point to `0x100`, not the garbage value at `0x120`! Likewise, any call to the stack that uses any memory addresses to the heap might need to be fixed before we can continue.

Therefore, we need to not only compact the heap, but also *relocate/forward* all existing references.