15-122: Principles of Imperative Computation

Recitation 22 Solutions — Josh Zimmerman

Modifying `heaps.c` to free non-empty heaps

In Huffman lab, we had to make sure that the priority queue was empty before we freed it. But what if instead we use a function pointer to let us free non-empty queues without leaking memory?

We need to modify our heaps to have a `elem_free` function as part of the data structure.

Let's write this now. Here's the start of the code for the new version of `pq_free`, and some modified code to support a function pointer.

```
1 typedef void (*elem_free_t)(elem);
2 struct heap_header {
3   int limit;              /* limit = capacity+1 */
4   int next;               /* 1 <= next && next <= limit */
5   elem* data;             /* \length(data) == limit */
6   elem_free_t elem_free; /* Non-NULL pointer to function that frees elems */
7 };
8 typedef struct heap_header* heap;
9
10 heap pq_new(int capacity, elem_free_t elem_free) {
11   REQUIRES(capacity > 0);
12   REQUIRES(elem_free != NULL);
13   heap H = xcalloc(1, sizeof(struct heap_header));
14   H->limit = capacity+1;
15   H->next = 1;
16   H->data = xcalloc(capacity+1, sizeof(elem));
17   H->elem_free = elem_free;
18   ENSURES(is_heap(H));
19   return H;
20 }
21
22 void pq_free(heap H) {
23   REQUIRES(is_heap(H));
24   for (int i = 1; i < H->next; i++) {
25       (*(H->elem_free))(H->data[i]) // Free each element of the heap
26   }
27   free(H->data);
28   free(H);
29 }
```