15-122: Principles of Imperative Computation

Recitation Week 8 — Nivedita Chopra

Visualizing the heap data structure

Use the visualization at `http://www.cs.usfca.edu/~galles/visualization/Heap.html` to insert the following elements into a min-heap, in the given order.

$$5, 3, 6, 7, 2, 6$$

Priority queue client and library interface

We use heaps to efficiently implement the *priority queue* interface.

```
1 /* Client interface */
2
3 // typedef _____________ elem;
4
5 // Returns true if e1 has strictly higher priority
6 bool higher_priority(elem e1, elem e2);
7
8 /* Library interface */
9
10 // typedef _____________ pq;
11
12 bool pq_empty(pq P);
13 bool pq_full(pq P);
14 pq pq_new(int capacity)
15   /*@requires capacity > 0; @*/
16   /*@ensures pq_empty(\result); @*/ ;
17
18 void pq_add(pq P, elem e)
19   /*@requires !pq_full(P); @*/ ;
20
21 elem pq_rem(pq P) // Removes highest priority element
22   /*@requires !pq_empty(P); @*/ ;
```

Checkpoint 0

Does this client interface define a min-heap or a max-heap, or something else?

```
1 typedef int elem;
2
3 bool higher_priority(elem x, elem y) {
4   return x > y;
5 }
```

Checkpoint 1

Define a client interface that ensures that, in a priority queue of strings, the *longest* strings always gets returned first.

Deletion of the lowest-priority element from a heap

```
1 elem pq_rem(heap H)
2 //@requires is_pq(H) && !pq_empty(H);
3 //@ensures is_pq(H);
4 {
5   int i = H->next;
6   elem min = H->data[1];
7   (H->next)--;
8
9   if (H->next > 1) {
10    H->data[1] = H->data[H->next];
11    sift_down(H);
12  }
13
14  return min;
15 }
```

Checkpoint 2

Complete the code for `sift_down`.

```
1 void sift_down(heap H)
2
3 //@requires ____________________________________________________________
4 //@ensures is_heap(H);
5 {
6   int i = 1;
7
8   while (_____________________________________________________)
9   //@loop_invariant 1 <= i && i < H->next;
10  //@loop_invariant is_heap_except_down(H, i);
11  //@loop_invariant grandparent_check(H, i);
12  {
13    int left = 2*i;
14    int right = left+1;
15
16    if (_____________________________________________________________________)
17      return;
18
19    if (_____________________________________________________________________) {
20      swap_up(H, left);
21      i = left;
22
23    } else {
24      //@assert ____________________________________________________________
25      swap_up(H, right);
26      i = right;
27    }
28  }
29
30  return;
31 }
```

Checkpoint 3

Check that the preconditions imply the loop invariants hold initially, and that they are satisfied when `sift_down` is called from `pq_rem`.

Show that the grandparent check is necessary as a loop invariant.

Prove that the loop invariants imply the postcondition for the return on line 17 and on line 32.