Recitation 10: Heaps of Fun — 22 March

Visualizing heaps

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.

```
/* Client interface */
// typedef _______ elem;
typedef void* elem;

// f(x,y) returns true if e1 is STRICTLY higher priority than e2
typedef bool has_higher_priority_fn(elem e1, elem e2);

/* Library interface */
//typedef ______* pq_t;

bool pq_empty(pq_t H);
bool pq_full(pq_t H);
pq_t pq_new(int capacity, has_higher_priority_fn* priority)
  /*@requires capacity > 0 && priority != NULL; @*/
  /*@ensures pq_empty(\result); @*/ ;

void pq_add(pq_t H, elem x)
  /*@requires !pq_full(H); @*/ ;

elem pq_rem(pq_t H) // Removes highest priority element
  /*@requires !pq_empty(H); @*/
```

Checkpoint 0

If the client's `elem` type is picked to be `void*`, will this client interface cause `pq_new(20, &higher_priority)` to return a min-heap, a max-heap, or something else?

```
1 bool higher_priority(void* x, void* y)
2 //@requires x != NULL && \hastag(int*, x);
3 //@requires y != NULL && \hastag(int*, y);
4 {
5   return *(int*)x > *(int*)y;
6 }
```

Checkpoint 1

Write a function (of type `has_higher_priority_fn`) that ensures that, in a priority queue of (pointers to) strings, the *longest* strings always gets returned first. The `string_length` function may be helpful.

Deletion of the highest-priority element from a heap

You may need the following functions: `is_safe_heap`, `ok_above`

```
1  void sift_down(heap* H)
2  //@requires ____________________________________________________________;
3  //@ensures is_heap(H);
4  {
5    int i = 1;
6    while (_____________________________________________________________)
7    //@loop_invariant 1 <= i && i < H->next;
8    //@loop_invariant is_heap_except_down(H, i);
9    //@loop_invariant grandparent_check(H, i);
10   {
11     int left = 2*i;
12     int right = left+1;
13     if (_____________________________________________________________)
14       return;
15     if (_____________________________________________________________) {
16       swap_up(H, left);
17       i = left;
18     } else {
19       //@assert ___________________________________________________________;
20       swap_up(H, right);
21       i = right;
22     }
23   }
24 }
25
26 elem pq_rem(heap* H)
27 //@requires is_heap(H) && !pq_empty(H);
28 //@ensures is_heap(H);
29 {
30   int i = H->next;
31   elem min = H->data[1];
32   (H->next)--;
33   if (H->next > 1) {
34     H->data[1] = H->data[H->next];
35     sift_down(H);
36   }
37   return min;
38 }
```

Checkpoint 2

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

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

(c) Prove that the loop invariants imply the postcondition for the return on line 14 and on line 23.