15-122: Principles of Imperative Computation

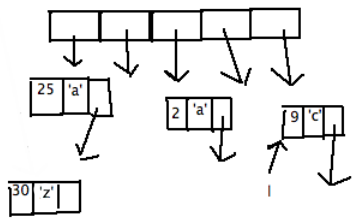
Recitation 14 Solutions — Josh Zimmerman

ht_insert

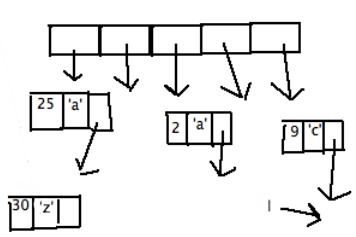
```
1 void ht_insert(ht H, elem e)
2 //@requires is_ht(H);
3 //@requires e != NULL; // We require that elem is a pointer type
4 //@ensures is_ht(H);
5 //@ensures ht_lookup(H, elem_key(e)) != NULL;
6 {
7    key k = elem_key(e);
8    int i = hash(k, H->capacity);
9
10   chain* p = H->table[i];
11   while (p != NULL)
12   //@loop_invariant is_chain(p, i, H->capacity);
13   {
14      //@assert p->data != NULL;
15      if (key_equal(elem_key(p->data), k)) {
16         /* overwrite existing element */
17         p->data = e;
18         return;
19      } else {
20         p = p->next;
21      }
22   }
23   //@assert p == NULL;
24   /* prepend new element */
25   chain* q = alloc(struct chain_node);
26   q->data = e;
27   q->next = H->table[i];
28   H->table[i] = q;
29   (H->size)++;
30   return;
31 }
```

Insert the key 14 with the value 'j' into the hash table pictured earlier in the handout, after updating the key '30' to 'z'. Start your diagrams after line 8 of the insert code.

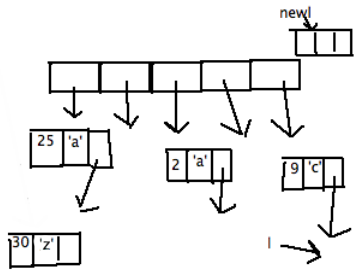
Solution: First we set `l` to point to the same thing that `H[4]` points to.



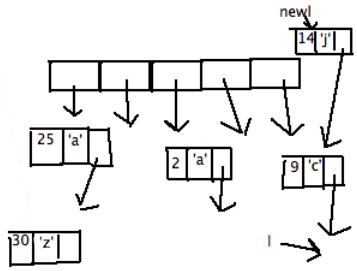
Next, we start to iterate, since 9 isn't the key we're looking for:



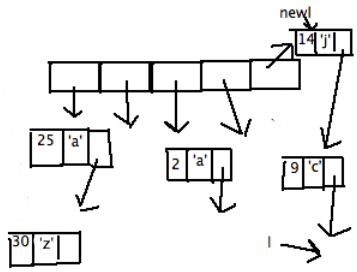
But now `l == NULL`, so we break out of the while loop. Next, we allocate space for a new `struct list_node`:



To save space, I'll do the next two steps in one diagram:



Next, we repoint `H->table[h]` and increment `H->n`, which I don't show here.



Hashtable lookup

Next, work together to write hashtable lookup code, using the following interface and struct definition of hashtables:

```
1 struct chain_node {
2   elem data;       /* data != NULL */
3   struct chain_node* next;
4 };
5 typedef struct chain_node chain;
6
```

```
7 struct ht_header {
8   int size;     /* size >= 0 */
9   int capacity;    /* capacity > 0 */
10  chain*[] table;  /* \length(table) == capacity */
11 };
12 typedef struct ht_header* ht;
13
14 /* ht_lookup(H, k) returns NULL if key k not present in H */
15 elem ht_lookup(ht H, key k)
16 //@requires is_ht(H);
17 ;
```

Solution:

```
1 elem ht_lookup(ht H, key k)
2 //@requires is_ht(H);
3 {
4    int i = hash(k, H->capacity);
5    chain* p = H->table[i];
6    while (p != NULL)
7    //@loop_invariant is_chain(p, i, H->capacity);
8    {
9       //@assert p->data != NULL;
10      if (key_equal(elem_key(p->data), k)) {
11         return p->data;
12      } else {
13         p = p->next;
14      }
15   }
16   /* not in chain */
17   return NULL;
18 }
```