Recitation 14, 10/17/12 15-122D Alex Cappiello

Consider the following client implementation:

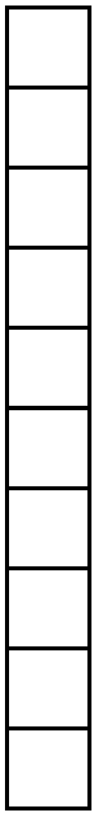
```
typedef struct foo* elem;
typedef int key;
struct foo {
  int key;
  int data;
};

int hash (key k, int m) {
  return k % m; // This is a terrible hash function.
}

bool key_equal (key k1, key k2) {
  return k1 == k2;
}

key elem_key (struct foo *x) {
  return x->key;
}
```

Suppose we're inserting the following (key, data) pairs in order:
(33, 54), (24, 77), (92, 12), (54, 31), (4, 11), (54, 31), (92, 92)
Draw the result of these insertions. (capacity = 10)



Suppose we want to look up an element in our hash table, but we don’t have its key. Instead, we have to do a lookup based on its data value. Building off of the previous example, the data value would be given by e->data. Assume you have the following:

```
  typedef int vtype;
  bool value_equal (vtype v1, vtype v2);
```

that correctly does what its name suggests. Fill in the following function that does the task described above.
(There is intentionally more space than is needed).

```
elem ht_search (ht H, vtype v)
//@requires

//@ensures

{









}
```