Hash Tables

LAST

Hashing

Genericity

TODAY

- Client vs library interface
- Implementing

  hash table dictionaries

NEXT

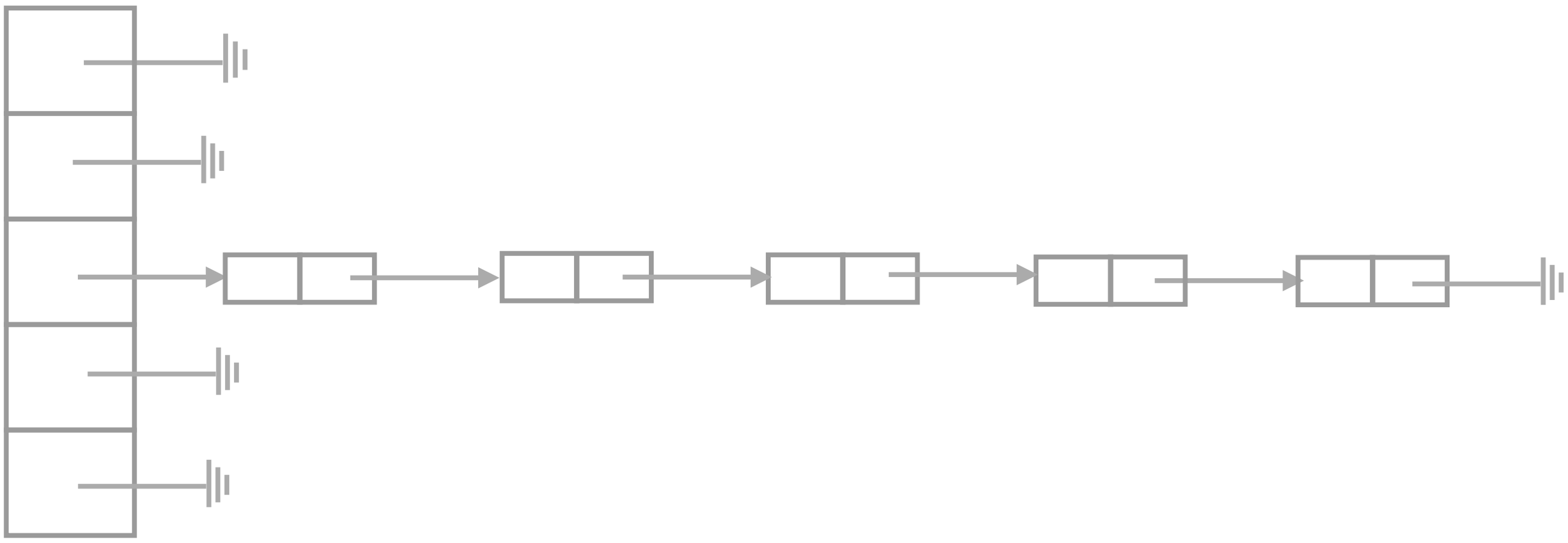
Making

code generic

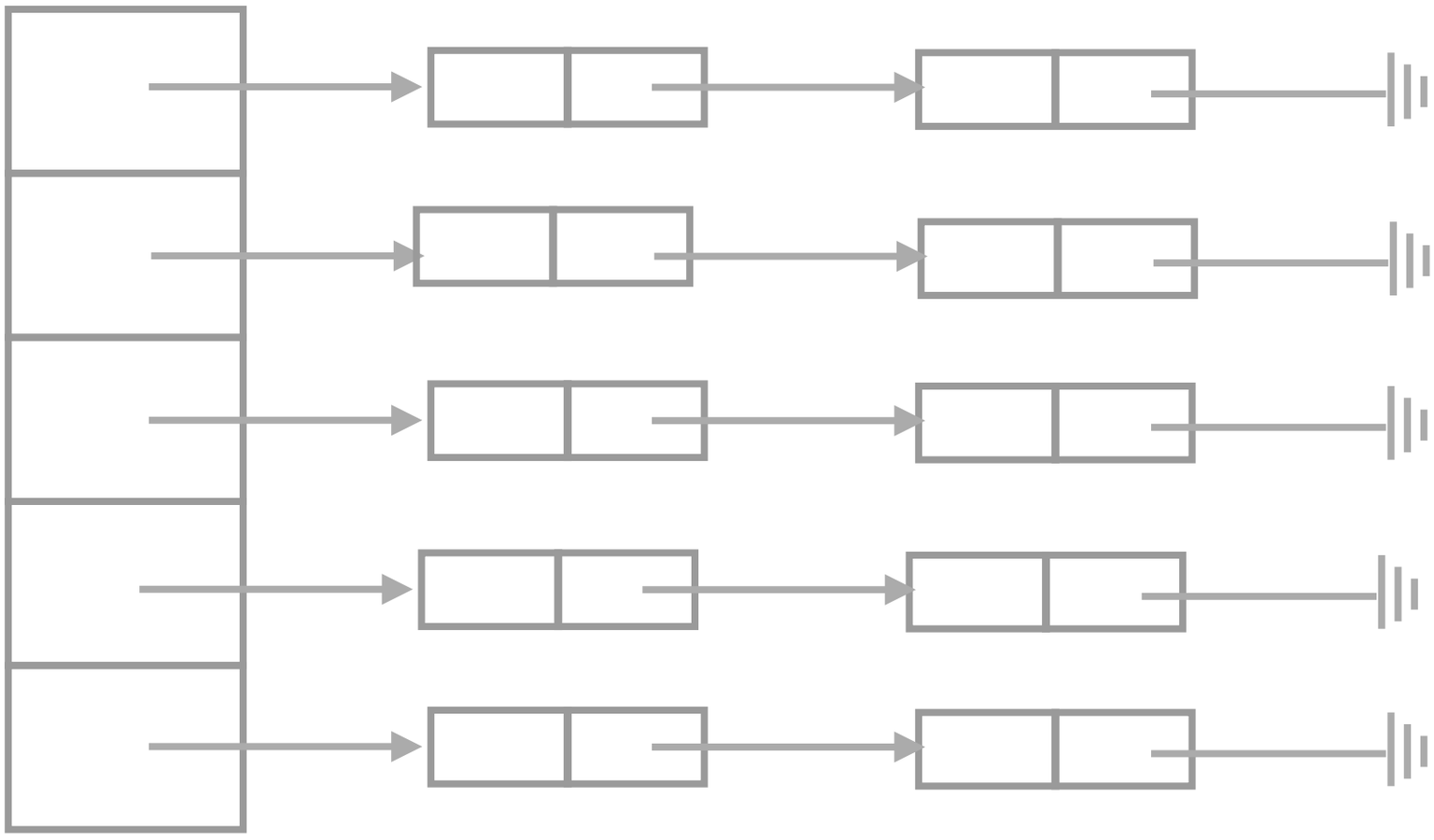
Hashing

m is the capacity of the table

Example: separate chaining



Example: separate chaining



Implementing dictionaries

	unsorted (key,value) array	(key, value) array sorted by key	linked list with (key,value) data	Hash tables
lookup	O(n)	O(log n)	O(n)	O(n/m) average O(1) average and amortized
insert	O(1) amortized	O(n)	O(1)	O(n/m) average O(1) average and amortized

Assuming keys are randomly distributed

Hashing fruit — You are the client

Entries you've allocated

0xA0:

--- "**apple**", 20, $3

0xB0:

--- "**banana**", 10, $2

0xC0:

--- "**pumpkin**", 50, $10

0xD0:

--- "**banana**", 20, $1

0xE0:

--- "**lemon**", 5, $3

Keys	Hash values
"**apple**"	-1290151091
"**berry**"	-514151789
"**banana**"	207055587
"**grape**"	-581390202
"**lemon**"	-665562942
"**lime**"	2086736531
"**pumpkin**"	-1189657311

Operations

1. insert entry 0xA0
2. insert entry 0xB0
3. insert entry 0xC0
4. lookup "**apple**"
5. lookup "**lime**"
6. insert entry 0xD0
7. insert entry 0xE0

Hashing fruit — You are the client

Entries you've allocated

0xA0:

--- "**apple**", 20, $3

0xB0:

--- "**banana**", 10, $2

0xC0:

--- "**pumpkin**", 50, $10

0xD0:

--- "**banana**", 20, $1

0xE0:

--- "**lemon**", 5, $3

Keys	Hash values
"**apple**"	-1290151091
"**berry**"	-514151789
"**banana**"	207055587
"**grape**"	-581390202
"**lemon**"	-665562942
"**lime**"	2086736531
"**pumpkin**"	-1189657311

Operations

1. insert entry 0xA0
2. insert entry 0xB0
3. insert entry 0xC0
4. lookup "**apple**"
5. lookup "**lime**"
6. insert entry 0xD0
7. insert entry 0xE0

Library Interface

```
// typedef ______* hdict_t;

hdict_t hdict_new(int capacity)
/*@requires capacity > 0; @*/
/*@ensures \result != NULL; @*/ ;

entry hdict_lookup(hdict_t H, key k)
/*@requires H != NULL; @*

void hdict_insert(hdict_t H, entry x)
/*@requires H != NULL && x != NULL; @*/
```

Client Interface

```
// typedef ______  key;
// typedef ______ entry;;

key  entry_key(entry x)
/*@requires x != NULL; @*/ ;

int  key_hash(key k);

bool key_equiv(key k1, key k2);
```

Library interface

```
// typedef ______* hdict_t;

hdict_t hdict_new(int capacity)
/*@requires capacity > 0; @*/
/*@ensures \result != NULL; @*/ ;

entry hdict_lookup(hdict_t H, key k)
/*@requires H != NULL; @*/

void hdict_insert(hdict_t H, entry x)
/*@requires H != NULL && x != NULL; @*/
```

```
// typedef ______* entry;
// typedef ______  key;

key  entry_key(entry x)
/*@requires x != NULL; @*/ ;

int  key_hash(key k);

bool key_equiv(key k1, key k2);
```

client-side types

What should the postconditions of **`lookup`** and **`insert`** be?

Library interface

```
// typedef ______* hdict_t;

hdict_t hdict_new(int capacity)
/*@requires capacity > 0; @*/
/*@ensures \result != NULL; @*/ ;
```

```
// typedef ______* entry;
// typedef ______  key;

key  entry_key(entry x)
/*@requires x != NULL; @*/ ;

int  key_hash(key k);

bool key_equiv(key k1, key k2);
```

```
entry hdict_lookup(hdict_t H, key k)
/*@requires H != NULL; @*/
/*@ensures \result == NULL || key_equiv(entry_key(\result), k); @*/ ;

void hdict_insert(hdict_t H, entry x)
/*@requires H != NULL && x != NULL; @*/
/*@ensures hdict_lookup(H, entry_key(x)) == x; @*/ ;
```

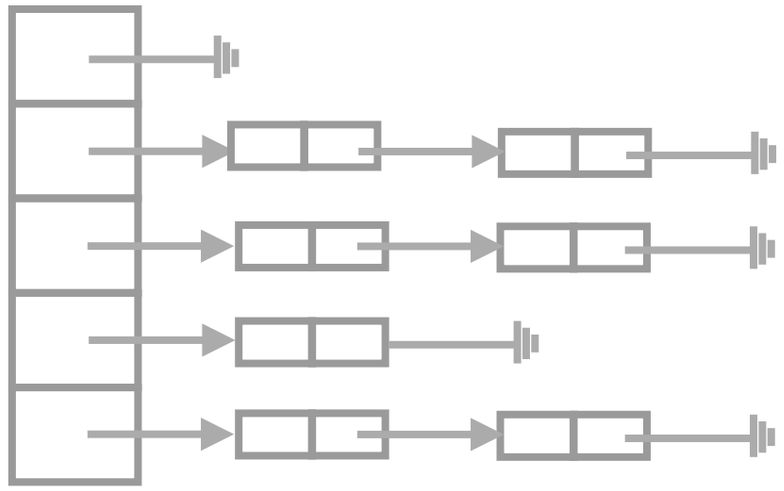
```
coin hdict.c0

hdict.c0:18.1-18.4:error:identifier 'key' at top level
```

Need to compile in this order:

```
cc0 -d lib/*.c0 hdict-client.c0 hdict.c0 main.c0
```

Library Implementation:

structure definitions, data structure invariants



```
typedef struct chain_node chain;

struct chain_node {
  entry  data;       // != NULL; contains both key and value
  chain* next;
};
```

```
typedef struct hdict_header hdict;

struct hdict_header {
  int size;                 // 0 <= size
  chain*[] table;           // \length(table) == capacity
  int capacity;             // 0 < capacity
};
```

Data (representation) structure invariant

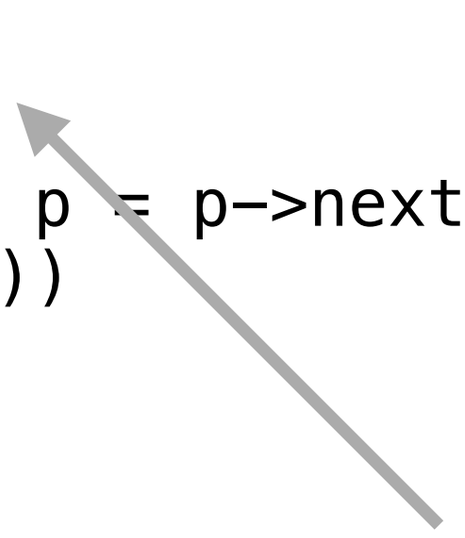
```
bool is_array_expected_length(chain*[] table, int length) {
  //@assert \length(table) == length;
  return true;
}
```

```
bool is_hdict(hdict* H) {
  return H != NULL
         && H->capacity > 0
         && H->size >= 0
         && is_array_expected_length(H->table, H->capacity);
    /* more properties */

}
```

```c
hdict* hdict_new(int capacity)
//@requires capacity > 0;
//@ensures is_hdict(\result);
{
  hdict* H = alloc(hdict);
  H->size = 0;
  H->capacity = capacity;
  H->table = alloc_array(chain*, capacity);
  return H;
}
```

```c
typedef struct chain_node chain;
struct chain_node {
  entry  data;
  chain* next;
};

typedef struct hdict_header hdict;
struct hdict_header {
  int size;
  chain*[] table;
  int capacity;
};
```

```
entry hdict_lookup(hdict* H, key k)
//@requires is_hdict(H);
//@ensures \result == NULL || key_equiv(entry_key(\result), k);
{
  int i = abs(key_hash(k) % H->capacity);

  for (chain* p = H->table[i]; p != NULL; p = p->next) {
      if (key_equiv(entry_key(p->data), k))
        return p->data;

  }
  return NULL;
}
```



Pull this out as a helper function

```
entry hdict_lookup(hdict* H, key k)
//@requires is_hdict(H);
//@ensures \result == NULL || key_equiv(entry_key(\result), k);
{
  int i = index_of_key(H, k);
  for (chain* p = H->table[i]; p != NULL; p = p->next) {
    if (key_equiv(entry_key(p->data), k))
      return p->data;
  }
  return NULL;
}
```

What should the contract of `index_of_key` be?

```
int index_of_key(hdict* H, key k)
//@requires is_hdict(H);
//@ensures 0 <= \result && \result < H->capacity;
{
  return abs(key_hash(k) % H->capacity);
}
```

```
void hdict_insert(hdict* H, entry x)
//@requires is_hdict(H);
//@requires x != NULL;
//@ensures is_hdict(H);
//@ensures x == hdict_lookup(H, entry_key(x));
{
  key k = entry_key(x);
  int i = index_of_key(H, k);
  for (chain* p = H->table[i]; p != NULL; p = p->next) {
    if (key_equiv(entry_key(p->data), k)) {
      p->data = x;
      return;
    }
  }
  // prepend new entry
  chain* p = alloc(chain);
  p->data = x;
  p->next = H->table[i];
  H->table[i] = p;
  (H->size)++;
}
```