# 15-122: Principles of Imperative Computation — Spring 2018

## Lab 9: Legacy of the void* — 1 March

**Collaboration:** In lab, we encourage collaboration and discussion as you work through the problems. These activities, like recitation, are meant to get you to review what we've learned, look at problems from a different perspective and allow you to ask questions about topics you don't understand. We encourage discussing problems with your neighbors as you work through this lab!

**Setup:** Copy the lab code from our public directory to your private directory:

```
% cd private/15122
% cp -R /afs/qatar.cmu.edu/usr4/tjabban/public/lab-rollcall .
% cd lab-rollcall
```

You should write your code in a new file, `rollcall.c1`, in the directory `lab-rollcall`.

**Grading:** Finish tasks (1.a), (1.b), and (1.c) for 2 points, and additionally finish (1.d) for 3 points.

## Using generic hash tables

In this lab, we'll be using the object-oriented tables discussed in lecture last week, but we'll be implementing a *dictionary* interface instead of the *set* interface.

```
/*** Client interface ***/

typedef void* key;
typedef void* value;

typedef bool key_equiv_fn(key x, key y);
typedef int key_hash_fn(key x);

/*** Library interface ***/

// typedef ______* hset_t;
typedef struct hset_header* hset_t;

hset_t hset_new(int capacity, key_equiv_fn* equiv, key_hash_fn* hash)
  /*@requires capacity > 0 && equiv != NULL && hash != NULL; @*/
  /*@ensures \result != NULL; @*/ ;

value hset_lookup(hset_t H, key k)
  /*@requires H != NULL; @*/ ;

void hset_insert(hset_t H, key k, value v)
  /*@requires H != NULL && v != NULL; @*/
  /*@ensures hset_lookup(H, k) == v; @*/ ;
```

Our sample application will be used in checking student attendance. Your code for this should go in the file `rollcall.c1`.

**(1.a)** Define a struct that represents students. Its fields should include `andrew_id` (**string**), `days_present` (**int**), and `days_absent` (**int**). You can include other fields if you want, but you need these fields with these types.

1pt

Write out the definition of this struct. Include a **typedef** so that you can allocate structs with **alloc**(student).

**(1.b)** Write client functions for a hashtable based on student information. For this lab we will be using pointers to **string**s (**string***) to represent the key, and pointers to students (student*) to represent the value.

*Hint:* Your functions should have the requirement that x and y are both non-NULL and have **string*** as their tag.

```
int hash_student(key x);
bool students_same_andrewid(key x, key y);
```

**(1.c)** Write a function that initializes a hset_t with students that have no attendance record. Don't worry about what happens if there are duplicates in this array.

```
hset_t new_roster(string[] andrew_ids, int len)
//@requires \length(andrew_ids) == len;
```

2pt

At this point, you should create a trivial main() function just to make sure your code compiles.

**(1.d)** Write functions that increment a student's attendance record and returns a student's attendance record.

```
void mark_present(hset_t H, string andrew_id)
//@requires H != NULL;

void mark_absent(hset_t H, string andrew_id)
//@requires H != NULL;
```

These functions should manipulate the days_present and days_absent fields stored in the hash table, so that hset_lookup can access these fields later on.

3pt

You can compile and run your code with test-rollcall.c1:

```
% cc0 -d hset.c1 rollcall.c1 test-rollcall.c1
% ./a.out
Enrolling bovik, rjsimmon, fp, and niveditc... done.
Student gburdell is not enrolled...
Student bovik is enrolled...
Student rjsimmon is enrolled...
Student twm is not enrolled...

Student bovik: 5 present, 4 absent...
Student rjsimmon: 8 present, 1 absent...
Student niveditc: 8 present, 1 absent...
Student fp: 2 present, 7 absent...
Done!
```