15-122: Principles of Imperative Computation

Lab 8: Legacy of the void* — Rob Simmons

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/andrew/course/15/122/misc/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.

```
1  /*** Client interface ***/
2
3  // typedef _______ key;
4  typedef string key;
5
6  // typedef ______* value;
7  typedef struct student_info value;
8
9  typedef bool key_equiv_fn(key x, key y);
10 typedef int key_hash_fn(key x);
11
12 /*** Library interface ***/
13
14 // typedef ______* hset_t;
15 typedef struct hset_header* hset_t;
16
17 hset_t hset_new(int capacity, key_equiv_fn* equiv, key_hash_fn* hash)
18   /*@requires capacity > 0 && equiv != NULL && hash != NULL; @*/
19   /*@ensures \result != NULL; @*/ ;
20
21 value hset_lookup(hset_t H, key k)
22   /*@requires H != NULL; @*/ ;
23
24 void hset_insert(hset_t H, key k, value v)
25   /*@requires H != NULL && v != NULL; @*/
26   /*@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) Represent students as a struct with fields `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.

Write a type definition so that you can allocate structs with `alloc(struct student_info)`.

(1.b) Write client functions for a hashtable based on student information. The hash function should create a hash value based only on the `andrew_id` string, and the equivalence function should check only the `andrew_id` fields for equality.

```
1 int hash_student(string x);
2 bool students_same_andrewid(string x, string 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.

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

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 attendence record and returns a student's attendence record.

```
1 void mark_present(hset_t H, string andrew_id)
2 //@requires H != NULL && hset_lookup(H, id) != NULL;
3
4 void mark_absent(hset_t H, string andrew_id)
5 //@requires H != NULL && hset_lookup(H, id) != 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.

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!
```