

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 generic hash tables library discussed in lecture last week:

```
1  /*****/
2  /*** Client interface ***/
3  /*****/
4  // typedef _____* elem;
5  typedef void* elem;
6
7  typedef bool elem_equiv_fn(elem x, elem y)
8    /*@requires x != NULL && y != NULL; @*/ ;
9
10 typedef int elem_hash_fn(elem x)
11    /*@requires x != NULL; @*/ ;
12
13 /*****/
14 /*** Library interface ***/
15 /*****/
16 // typedef _____* hset_t;
17 typedef struct hset_header* hset_t;
18
19 hset_t hset_new(int capacity, elem_equiv_fn* equiv, elem_hash_fn* hash)
20    /*@requires capacity > 0 && equiv != NULL && hash != NULL; @*/
21    /*@ensures \result != NULL; @*/ ;
22
23 elem hset_lookup(hset_t H, elem x)
24    /*@requires H != NULL && x != NULL; @*/ ;
25
26 void hset_insert(hset_t H, elem x)
27    /*@requires H != NULL && x != NULL; @*/
28    /*@ensures hset_lookup(H, x) == x; @*/ ;
```

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 refer to the structs as `stu` and allocate them with `alloc(stu)`.

(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(void* x)
2  //@requires x != NULL && \hastag(stu*, x);
3
4  bool students_same_andrewid(void* x, void* y)
5  //@requires x != NULL && \hastag(stu*, x);
6  //@requires y != NULL && \hastag(stu*, y);
```

(1.c) Write a function that instantiates 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 a function that makes it easier to access the hashsets created by `new_roster` by creating a dummy student struct with the right name, casting it to `void*`, and using it to look up the appropriate student in the hash table.

```
1  stu* lookup(hset_t H, string andrew_id)
2  //@requires H != NULL;
```

The pointer this function returns should be the one stored in the hash set, so that you could manipulate its `days_present` and `days_absent` fields and then look up those changes later when you look up the same `andrew id`.

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!
```