15-122: Principles of Imperative Computation

QuickCheck 5 — March 22, 2013

This QuickCheck will be conducted towards the end of recitation. You will have fifteen minutes to do this. Your TA will go over answers at the end.

Name:

Andrew ID:

Section (circle one): A B C D E F G H

Congratulations! *Simplicio* and *Sagredo* (you) have landed a summer internship at Facegook's internal tech division. And, by a stroke of good luck you are both on the same team!

Your project involves dealing with a lot of employee data, so you need a way to be able store employee records, so that you can access them in constant time. Your mentor, *Salviati* has already created a generic hash table library, which allows a value *elem* to be stored based on it's unique *key*.

In order to use the hash table, you need to provide the client code. The employee information is represented by the following struct -

```
1 struct employee_info{
2    int emp_id;
3    string first_name;
4    string last_name;
5    int wage;
6 };
7 typedef struct employee_info* employee;
```

It's obvious that **employee**s will be stored as values in the hash table. Given the fields above, what do you think is the appropriate key?

_____

Now fill in the required information below that the client (you) need to provide the library -

```
1 //set elem to employee
2 typedef _____________________;
3
4 //set the key type
5 typedef __________;
6
7 key elem_key(elem e){
8    ___________________________________
9 }
10
11 bool key_equal(key k1, key k2){
12    ____________________________
13 }
```

Now you need to do some analysis on the pay that employees are receiving. *Salviati* gives you an array of n **employees** sorted by **wage**. The wages are unique. You decide to create a Binary Search Tree of the data, for use in the analysis.

Again, **elem** is **employee**. But the **key** might not be the same. What is the appropriate **key** for the BST?

_____

This is what *Simplicio* has come up with for creating the BST -

```
1 bst make_tree(employee[] E, int n){
2    bst B = alloc(struct bst_header);
3    for(int i = 0; i < n; i++){
4       bst_insert(B, E[i]);
5    }
6    return B;
7 }
```

What's a problem with this code?

Fill in the following code that returns a BST close to the optimum case -
(Hint: Keep in mind the algorithm for binary search and remember that the array is sorted by **wage**)

```
1 bst make_tree(employee[] E, int n){
2    bst B = alloc(struct bst_header);
3    B->root = create(__________);
4    return B;
5 }
6
7 tree* create(employee[] E, int lower, int upper){
8    if(lower == upper) return ___________;
9    int mid = ____________________________________________
10   tree* T = alloc(struct tree_node);
11   T->data = _____________
12   T->left = ____________________________________
13   T->right = ____________________________________________
14   return T;
15 }
```

Now, you've been told that your actual assignment involves finding the employee with the lowest wage in the company. Using the BST would involve $O($_____$)$ time. So, it's better to use a priority queue that can do it in $O(\_\_)$ time.

Assume that you have created a priority queue H which has **wage** as the **key** and **employee** as the **elem**.

You are given the state legal minimum wage *min_wage*. You need to write a function that raises the pay of any employees whose current pay is below the legal minimum wage to the legal minimum wage.

e.g If the wages of employees A, B, C and D are (3, 1, 4, 2) and the *min_wage* is 3, after you run your function, wages should be (3, 3, 4, 3) respectively.

Fill in the the blanks below to create the required function.

```
1  void adjust_wages(heap H, int min_wages){
2     while(_____){
3        //remove the element with the minimum wage
4        employee temp = _____
5        //replace temp's wage with the minimum wage
6        _____
7        //insert temp back into the heap
8        _____
9     }
10 }
```