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 -

```c
struct employee_info{
    int emp_id;
    string first_name;
    string last_name;
    int wage;
};
typedef struct employee_info* employee;
```

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

```
typedef ______________;

key elem_key(elem e){
    ________________________
}

bool key_equal(key k1, key k2){
    ________________________
}
```

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

```c
typedef ______________;

//set the key type
typedef ______________;

//set elem to employee
typedef ______________;

key elem_key(elem e){
    ________________________
}

bool key_equal(key k1, key k2){
    ________________________
}
```
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 -

```c
bst make_tree(employee[] E, int n){
    bst B = alloc(struct bst_header);
    for(int i = 0; i < n; i++){
        bst_insert(B, E[i]);
    }
    return B;
}
```

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)

```c
bst make_tree(employee[] E, int n){
    bst B = alloc(struct bst_header);
    B->root = create(__________);
    return B;
}
```

```c
tree* create(employee[] E, int lower, int upper){
    if(lower == upper) return __________;
    int mid = __________;
    tree* T = alloc(struct tree_node);
    T->data = __________;
    T->left = __________;
    T->right = __________;
    return T;
}
```
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 $\text{min\_wage}$. 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 $\text{min\_wage}$ 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.

```c
void adjust_wages(heap H, int min_wage){
    while( ){
        //remove the element with the minimum wage
        employee temp = _____________;
        //replace temp’s wage with the minimum wage
        _____________;
        //insert temp back into the heap
        _____________;
    }
}
```