COSC 301: Operating Systems - Lab 2

For this lab, you will get some more practice with C by writing 3 different functions. There is some template code supplied in this repo to help get you started, and for testing your functions.

When you're finished, you should demo your code to me.

---

1. Write a C function called `string_in` that takes two `char *` as arguments. If the second string is contained within the first string, the function should return the address at which the contained string begins. The match should be done in a case-insensitive way. For example, `string_in("hats!", "AT")` should return the address of `'a'` in `"hats!"`. If the second string is not contained within the first string, the function should just return `NULL`. For this problem, you cannot use the built-in `strcasestr` function (since you are basically reimplementing it). You can, however, use any other built-in function. In particular, `strncasecmp` may be helpful.

2. Assume that you have the following struct declaration:

```
struct record {
    char name[128];
    unsigned char age;
    unsigned char shoe_size;
};
```

Write a `search_by_name` function that takes an array of `struct record`, the length of the array, a string to search for, and an index in the array to start the search from. The function should return the first index in the array in which the name in the record includes the search string (anywhere in the string), or -1 if there is no match found. The string match should be done in a case-insensitive way. You can use your `string_in` function from problem 1, but you cannot use `strcasestr` or `strncasecmp` or similar functions for this problem.

The function header should just be:

```
int search_by_name(struct record records[], int num_records,
                   const char *str, int start_index);
```

(Note: the lack of definite size of the array variable records is ok in C. In this case, we have to assume some additional input to tell us how many records-long the array is. Coincidentally, it's the same for C strings, too: the header above could have said "char name[]" for the name to search for. In that case, we'd have to assume that the C string is properly NULL-terminated.)

3. Write a `get_matches` function that uses the `search_by_name` function above. The function should take an array of `struct record`, an integer indicating the size of the array, and a search string, and return an array of integers that includes *all* the indexes in the array that match the given search string. The array of integers returned should contain, as the first element, the number of matches. Thus, the array size should always be *at least* 1.

The function prototype should be:

```
int *get_matches(struct record records[], int num_records, const char *str);
```