

CS107 Quiz 4 Solution

This is a closed book, closed note, closed computer quiz. You have 10 minutes to complete it.

SUNet ID (@**stanford.edu**): _____

First and Last Name: _____

I accept the letter and spirit of Stanford's Honor Code.

[signed] _____

**Do not turn over until
instructed to do so.**

Problem 1: assign2 Takeaways [6 points total, 2, 2, and 2]

Your `scan_token` function from `assign2` is well exercised by the following test harness, extracted verbatim from the assignment handout with expected output presented on the right.

```
const char *input = "super-duper-awesome-magnificent";
char buf[10];
const char *remaining = input;
while (scan_token(&remaining, "-", buf, sizeof(buf))) {
    printf("Next token: %s\n", buf);
}
```

```
Next token: super
Next token: duper
Next token: awesome
Next token: magnifice
Next token: nt
```

- a. Explain why the fourth printed token is "**magnifice**". In particular, why is it of length 9?

`scan_token` can read in a maximum of nine `chars`, because the 10th byte in `buf` is needed for the terminating null byte.

- b. Would replacing `&remaining` with `&input` as the first argument to `scan_token` produce the same output, assuming no other changes? Explain briefly.

Yep, same output! `remaining` and `input` each store the same address when execution reaches the `while` loop, so `&remaining` and `&input` are each two hops away from the same string constant.

- c. Briefly explain why the test harness breaks if `char buf[10];` is replaced by `char *buf;` while everything else remains the same.

If `buf` is an uninitialized `char *`, it doesn't point to valid memory. Scanning into memory it supposedly points to results in undefined behavior. (`sizeof(buf)` goes from 10 to 8, but that isn't why it breaks.)

Problem 2: General Memory Tomfoolery [4 points total, 2 and 2]

For each snippet, print the expected output if there are no memory errors. Otherwise, identify the first error you find and explain what's wrong. Present answers to the right of the code.

- a.

```
int ghost[] = { 0, 7, 13, 21, 31, 404, 1300, 1313 };
int *broom = &ghost[1];
int *witch = broom + 4;
printf("%d %d\n", witch[1], ghost[witch - broom]);
```

```
witch[1] is ghost[6]
witch - broom is 4
```

What's printed?
1300 31

- b. Assume all allocations succeed and that `realloc` always moves the allocated memory to a new address.

```
char *cardinal = strdup("under");
char *tree = cardinal + 5;
cardinal = realloc(cardinal, 14);
strcpy(tree, "graduate");
printf("%s\n", cardinal);
free(cardinal);
```

Nothing is printed, because at the time `strcpy` is called, `tree` points into memory no longer allocated.