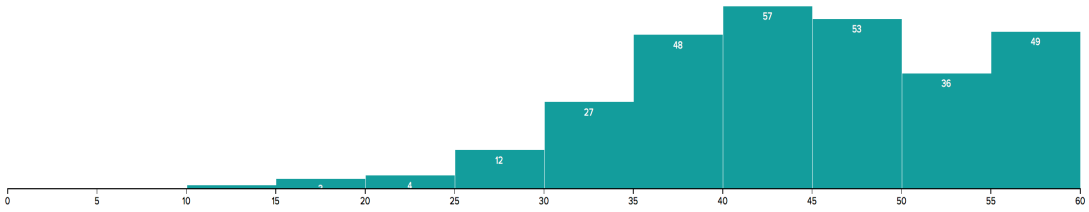
CS107 May 14, 2018
M Chang & J Zelenski

Midterm Solution



MINIMUM	MEDIAN	MAXIMUM	MEAN	STD DEV
13.0	44.0	60.0	43.86	9.72

You should receive email at your sunet@stanford.edu address notifying you of an exam grade on Gradescope. Above is the histogram and class statistics. A most lovely result all around, including a large group of perfect and near-perfect scores. ¡Felicidades!

CS107 is packed with challenging material and we ask a lot of you. No one escapes unscathed—the autograder runs roughshod over your code and we mark up your work with endless critique. Our red ink reflects the intricacies of systems and that there is always room for improvement, but we also want to assure you that your achievements in mastering the core content are being recognized and we will have that in mind when setting course grades. The median course grade for CS107 generally lands in the neighborhood of B+/B.

The midterm serves as a mid-quarter assessment. Bravo to those who achieved the result you worked toward! If your score shows you don't yet have a solid grip on the material or were not able to successfully demonstrate your ability on the exam, now is the time to figure out what changes to make going forward. I wrote up some broad advice intended to be generally useful at `http://cs107.stanford.edu/advice/midquarter.html`. Please reach out to us in office hours if you need further consultation on plotting a strategy for your specific situation.

Grading rubrics
Our rubrics are designed to apply constructively. We break down the problem into component tasks, each of which has an associated point value. Your score starts at zero. For each task that you correctly accomplish, you earn the points designated for that assessment. If the task was not attempted or incorrect, there is no change in score.

In the coding questions, care in handling pointers/memory, applying a necessary typecast, computing the right offset, and accessing at the correct level of indirection were the key skills being assessed. The bulk of the points are apportioned for those issues.

Regrades: If you believe there was an error in scoring your exam, please send email to cs107@cs with an explanation of the concern and ask for our review. The entire exam will be re-evaluated. Regrade requests must be initiated by Mon May 21st.

Problem 1: Bits, bytes, and numbers

J
M
B

Problem 2: C-strings

```
char *rotate(const char *s, char ch)
{
    char *first = strchr(s, ch);
    if (!first || first == s) return NULL;
    char *result = malloc(strlen(s)+1);
    strcpy(result, first);
    return strncat(result, s, first - s);
}
```

The `strcpy` call wrote the initial null terminator, the `strncat` moved it.

Problem 3: Pointers and generics

3a)

```
void *ith = (char *)base + i*width;
if (!sel || sel(ith))
     memcpy(ith, pval, width);
```

3b)

```
bool up(const void *p)
{
    return isupper(*(const char *)p);
}

void censor(char *s)
{
    replace(s, strlen(s), 1, up, "#");
}
```

3c)

```
replace(argv+1, argc-1, sizeof(argv[0]), NULL, &argv[1]);
```

Commentary from grading problem 1
This was an easy one for you – you have your bitwise ops down cold! Just FYI: we used Gradescope's "AI-enhanced" grading tool to batch grade these. If it mis-identified your answer, please let us know.

Commentary from grading problem 2
We were pleased to see how many of you turned to the string library to find the character and copy characters rather than re-implement that functionality! Be sure to keep your eye on the null terminator that is essential to a C-string. Allocate space for it and be sure to properly set it. A pointer to a single character is not the same thing as a one-character string unless you've put a null terminator to follow. The functions `strcat`/`strncat` don't so much add a null terminator as move the one that was already there.

Commentary from grading problem 3
This was an homage to the time invested in lab/assign 3 and 4. Much of the grading for this question hinged on handling `void*` pointers at the appropriate level of indirection. For example, the pointers exchanged with `replace` (arg to callback, replacement value) are all expressed as pointers to elements. Right versus wrong may come down to the presence or absence of a single `*` or `&`, but don't dismiss these issues as minor. This was a crucial insight being tested and it was heavily weighted in assessment. Keeping your pointers straight is challenging; those of you were vigilant should feel well-rewarded for your efforts!