

CS107 Lecture 11: Two-Minute Etudes

Problem 1: Arrays, Pointers, Arithmetic, and Strings

Consider each of the following code snippets, all of which compile and run to completion sans drama. What gets printed? You may assume all pointers are 8 bytes and that all allocation requests succeed.

a.

```
char alpaca[] = "llama";
char *sheep = alpaca;
sheep++;
sheep[2] = 'd';
sheep++;
printf("%s %s\n", alpaca, sheep);
```

b.

```
char oomph[] = "pizzazz";
char *flair = strchr(oomph, 'z');
char *panache = strrchr(oomph, 'z');
size_t diff = panache - flair;
printf("%zu %s\n", diff, oomph + diff);
```

c.

```
int absurd[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
int *silly = &absurd[3];
int *strange = &silly[5];
printf("%d %d\n", *strange, absurd[strange - silly]);
```

d.

```
char protein[] = "swordfish";
char *entree = protein;
char **plate = &entree;
printf("%zu %zu %zu %zu\n", sizeof(protein), sizeof(entree),
      sizeof(plate), sizeof(**plate));
```

e.

```
char furball[] = "meow";
char *feline = strdup(furball);
char *tabby = furball;
furball[0] = 'Y';
tabby[0] = feline[2] = 'w';
printf("%s %s\n", furball, feline);
free(feline);
```

Answers:
a. llama da
b. 4 pizzazz
c. 10 13
d. 16 16 16 16
e. Ywew

Problem 2: malloc and free Maladies

Consider each of the **malloc**, **strdup**, and **free**-centric code snippets, all of which compile and run at least in part if not in full. For those that run to completion, print the expected output. For those that include faulty code, assume the program crashes at the first invalid memory access, but print everything that would have printed out before the crash, and then explain the error. You can assume all memory allocations (here, and in Problem 3) succeed, so there's no need to call **assert**. Just write what you need to write to the right of the code.

a.

```
char *magician = strdup("abracadabra");
char *assistant = magician + 4;
strcpy(assistant, "cadaver");
printf("%s\n", magician);
free(assistant - 4);
```

b.

```
char *frosty = strdup("snowman");
char *olaf = frosty;
olaf[1] = 'h';
free(olaf);
printf("%s\n", frosty);
```

c.

```
char *wizard = strdup("sorcerer");
char *apprentice = wizard;
char *mystery = strdup(wizard);
strcpy(mystery + 3, "bet");
free(apprentice);
printf("%s\n", mystery);
free(mystery);
```

Answers:
a. abracadaver
b. No output, since printf accesses memory freed through olaf.
c. sorbet

Problem 3: Sizing Everything Correctly

What should the argument to **malloc** be in the following code snippet so that the allocation has the exact number of bytes needed to work correctly?

```
char *ghost = "boo";
char *comeback = malloc(_____);
strcpy(comeback, ghost);
strcat(comeback, "merang");
printf("%s\n", comeback);
free(comeback);
```

Answer: strlen(ghost) + strlen("merang") + 1, or simply 10. Prints boomerang.