15-122: Principles of Imperative Computation

Recitation 20 Solutions   Josh Zimmerman

What's wrong with this code?

```
1  // We're returning the address of a local variable, so the value of x
2  // may change unexpectedly if another function's local variables are in the
3  // wrong place on the stack.
4  int *add_dumb(int a, int b) {
5      int x = a + b;
6      return &x;
7  }
```

```
1  // A + i implicitly actually adds i * sizeof(int) to A. This is so that
2  // A[i] is the same as *(A + i).
3  // The loop upper bound should just be 10.
4  int main () {
5      int *A = xcalloc(10, sizeof(int));
6      for (int i = 0; i < 10 * sizeof(int); i++) {
7          *(A + i) = 0;
8      }
9      free(A);
10     return 0;
11 }
```

```
1  // When we call a function we make a copy of the argument, so add_one is
2  // modifying its own copy and the copy in main is unchanged.
3  void add_one(int a) {
4      a = a + 1;
5  }
6  int main() {
7      int x = 1;
8      add_one(x);
9      printf("%d\n", x);
10     return 0;
11 }
```

```
1  // "x = 1" assigns x to 1. In C, the expression evaluates to 1, since that's
2  // what we assigned x to. Any non-zero value is true, so we print out "woo\n"
3  int main() {
4      int x = 0;
5      if (x = 1) {
6          printf("woo\n");
7      }
8      return 0;
9  }
```

```
1  // s is not properly NUL-terminated, so when we print the string we have
2  // undefined behavior
3  int main() {
4      char s[] = {'a', 'b', 'c'};
5      printf("%s\n", s);
6      return 0;
7  }
```

```
1  // strlen(y) doesn't count the null-terminator, so we copy at most 6 characters
2  // into x. Since malloc doesn't intialize memory to 0, this means that x
```

```
 3 // might not be NUL—terminated, and so trying to get the length of x is
 4 // undefined behavior
 5 int main () {
 6     char *y = "hello!";
 7     char *x = xmalloc(7 * sizeof(char));
 8     strncpy(x, y, strlen(y));
 9     printf("%zu\n", strlen(x));
10     free(x);
11     return 0;
12 }
```

```
 1 // We're freeing something we never malloc'd, which we should never do —— only
 2 // free something you malloc'd. This gives undefined behavior as—is.
 3 int foo(char *s) {
 4     printf("The string is %s\n", s);
 5     free(s);
 6 }
 7 int main() {
 8     char *s = "hello";
 9     foo(s);
10     return 0;
11 }
```