15-213 Malloc Recitation

Waiting for SeanSPS (purple) to make a move.

Map: DataBus by TeamBS
comrade14ck (green) was randomly chosen to go first
YOU were cornered by SeanSPS (purple). You may still win if your score is highest in the end
fragmer2k: gah!
comrade14ck: pwned
SeanSPS: the real battle starts now!

>!
Allocate The Right Amount of Space
Which of these are correct, incorrect, or incorrect but will still behave as expected?

- int *x = malloc(sizeof(int *));
- int *x = malloc(sizeof(int));
- int *x = malloc(sizeof(x));
- int *x = malloc(sizeof(*x));
- char *orig = “some string”;
  - char *copy = malloc(orig);
  - char *copy = malloc(strlen(orig));
  - char *copy = malloc(strlen(orig + 1));
  - char *copy = malloc(strlen(orig) + 1);
- struct player {
    int health;
    char name[4];
}
- struct player *p = malloc(sizeof(struct player *));
- struct player *p = malloc(sizeof(struct player));
What's wrong with this code?

```c
#define SUCCESS 0
#define ERROR (-1)

typedef struct {
    char *name;
    unsigned age;
    double height;
} person_t;

int person_init(person_t *p) {
    p = malloc(sizeof(person_t));
    p->name = NULL;
    p->age = 18;
    p->height = 5.5;
    return SUCCESS;
}
```
What's wrong with this code?

```c
person_t *new_person() {
    person_t *p = malloc(sizeof(person_t));
    if (person_init(p) != SUCCESS) {
        return NULL;
    }
    return p;
}
```
static person_t *people;

int init_people(int n) {
    people = malloc(sizeof(person_t) * n);
    if (!people)
        return ERROR;
    for (int i = 0; i < n; i++)
        person_init(&people[i]);
    return SUCCESS;
}

int clear_person(int i) {
    return person_init(&people[i]);
}
What's wrong with this code?

```c
int main() {
    pid_t pid;
    if ((pid = fork()) != 0) {
        int *status;
        waitpid(pid, status, 0);
        printf("Child %d is done!\n", pid);
    } else {
        //Really long computation
    }
    return 0;
}
```
int main() {
    int *counts = malloc(MAX * sizeof(int));
    while (1) {
        int n;
        printf("Enter a number:\n");
        scanf("%d", &n);
        if (n >= 0 && n < MAX) {
            counts[n]++;
        } else if (n == -1) {
            break;
        }
    }
    int i;
    for (i = 0; i < MAX; i++) {
        printf("%d appeared %d times\n", i, counts[i]);
    }
    return 0;
}
Things to Remember

- Allocate the right amount of space: if you are allocating something to put in a “person *”, allocate sizeof(person), etc...
- When a function takes in a pointer to an object, it's wrong to allocate space for that object inside the function.
- ALWAYS check the return value of malloc.
- You should call free() on every pointer you get back from malloc exactly one, once you're done using it.
- Don't overstep your bounds.
- Pointers don't magically point to something, you have to allocate space for an object to point to.
- malloc() doesn't initialize the returned memory – use calloc() if appropriate. See the (m|c|re)alloc manpages for exact behavior.
Macros in C

- A macro is a code fragment that has been given a name.
- The preprocessor will go through your source and replace every occurrence of that name with the fragment of code.
- Macros can make your code cleaner, and yet not incur the overhead of a function call.
- How (not) to use macros…
What’s wrong here?

```
#define twice(x) 2 * x
twice(x + 1) = 2x + 2?
```

```
#define twice(x) x + x
#define min(X, Y) ((X) < (Y) ? (X) : (Y))
twice(x++) = 2x?
min(a, b++)?
min(foo(a), foo(b))?```
#define debug_printf( is_debug, str ) \
   if ( is_debug ) printf( "%s\n", str )

if (x < 0) debug_printf(debug_on, "Negative input");
else debug_printf(debug_on, "Non-negative input");
if (x < 0)
   if (debugon) printf("%s\n", "Negative input");
   else if (debugon) printf("%s\n", "OK input");

"OK input" never prints!
Things to Remember

- Surround names in macros with parentheses
- Don’t pass code with side effects to macros (you have no idea how many times they’re evaluated)
- Try not to evaluate macro arguments more than once in your macros
- When using macros in conditionals, put braces around them