## Header guards

- When making our own `.h` files, we will include *header guards*
- Header guards useful for the case where your header contains *definitions*. It prevents definition duplications (which give compiler errors) when multiple `.c` files include the same `.h` file.
- This will be clearer when we discuss `structs`; for now, follow the pattern in the following example:

## Header guards

```c
// rectangle.h:
// include lines like this at the top; change the all-caps
// name to match the file name, rectangle.h in this case
#ifndef RECTANGLE_H
#define RECTANGLE_H

struct rectangle {  //you don't need to know what this is yet
    int height;
    int width;
};

// include this line at the bottom
#endif
```

added
header
guards

## Recall: `sizeof`

- How big is an `int` (on ugrad)?
- `sizeof` operator returns size in bytes

```
// sizeof_eg_1.c:
#include <stdio.h>
int main() {
    int int_bytes = sizeof(int);
    printf("# bytes in int = %d", int_bytes);
    return 0;
}
```

```
$ gcc sizeof_eg_1.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
# bytes in int = 4
```

## Recall: `sizeof`

```c
// sizeof_eg_2.c:
#include <stdio.h>

int main() {
    printf("# bytes in char = %lu\n", sizeof(char));
    printf("# bytes in int = %lu\n", sizeof(int));
    printf("# bytes in float = %lu\n", sizeof(float));
    printf("# bytes in double = %lu\n", sizeof(double));
    return 0;
}
```

```
$ gcc sizeof_eg_2.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
# bytes in char = 1
# bytes in int = 4
# bytes in float = 4
# bytes in double = 8
```

## sizeof with Arrays

return type of sizeof operator is size_t which is a long unsigned

How big is an array?

this only works if we are initializing all elements with 0

```
// sizeof_eg_3.c:
#include <stdio.h>

int main() {
    int days[30] = {0}; // initializes all elements to 0
    printf("# bytes in days array = %lu\n", sizeof(days));
    return 0;
}
```

```
$ gcc sizeof_eg_3.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
# bytes in days array = 120
```

4 bytes per int, 30 ints

## sizeof with Strings

How big is a string?

```
// sizeof_eg_4.c:
#include <stdio.h>

int main() {
    char pet[] = "cat";
    printf("# bytes in pet string = %lu\n", sizeof(pet));
    return 0;
}
```

use double quotes for initializing a string

```
$ gcc sizeof_eg_4.c -std=c99 -pedantic -Wall -Wextra
$ ./a.out
# bytes in pet string = 4
```

1 byte per character, 4 characters (including terminator)

recall that the last character of a string is always the null terminator (i.e., '\0')