Header Files

Matthew Mussomele

So Far

So far you have only worked on tasks that required you to fill out a single file.

While only one file is needed for small projects, medium and large projects often require several to dozens to hundreds of files.

These files need some way of knowing what each other can do.

Header Files

Header files are C++'s way for files to communicate to each other what they can do.

You may have noticed files ending with the ".h" extension in the tasks you've received so far.

Those were header files, and they tell the testing files what functions are available for testing.

Header Files

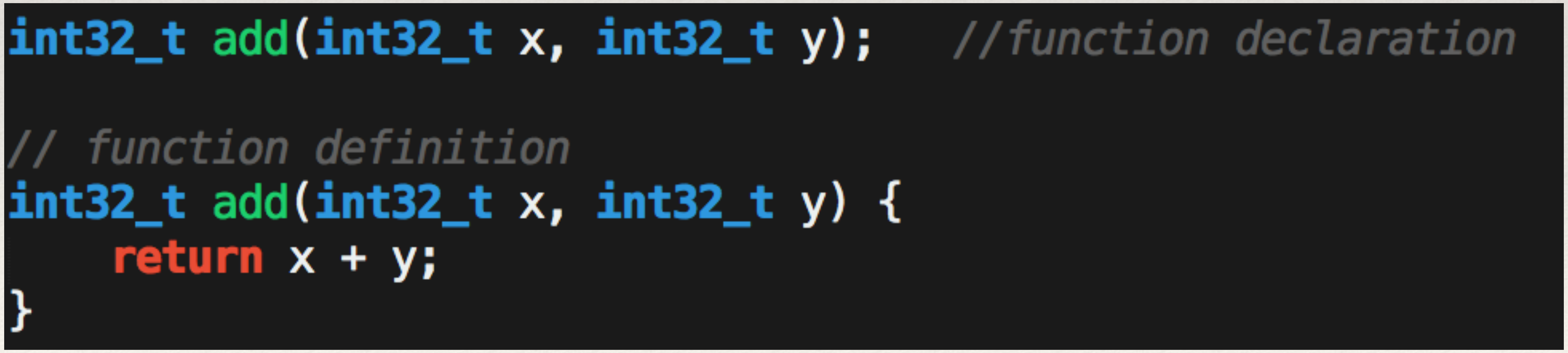
Header files can have as little as variable or function declarations.

They can also have variable instantiations or function definitions but this is not required.

Filling Out Header Files

A header file must have declarations for all functions that are going to be used in other files.

A function declaration is just like a definition, but without the body.



```
int32_t add(int32_t x, int32_t y);    //function declaration

// function definition
int32_t add(int32_t x, int32_t y) {
    return x + y;
}
```

Using Header Files

Simply fill a header file with declarations for all the functions that you plan to use in another file.

After you have done so, import the header file into the other file using:

```
#include “my_header.h”
```

Now, when compiling that file, C++ knows that all the functions declared in the header file exist.

Compiling With Header Files

Importing a header file in a C++ source file will not automatically grab it's source code counterpart.

You must still compile the two files together in order for it to work.

Say that we have file1.cpp and file2.cpp. If file2.cpp uses functions from file1.cpp, the command we would use to compile them is:

```
g++ file1.cpp file2.cpp -o result_file
```

Safeguards in Header Files

Header files should have some safeguards against dangerous import behaviors.

Imagine two files that import each other.

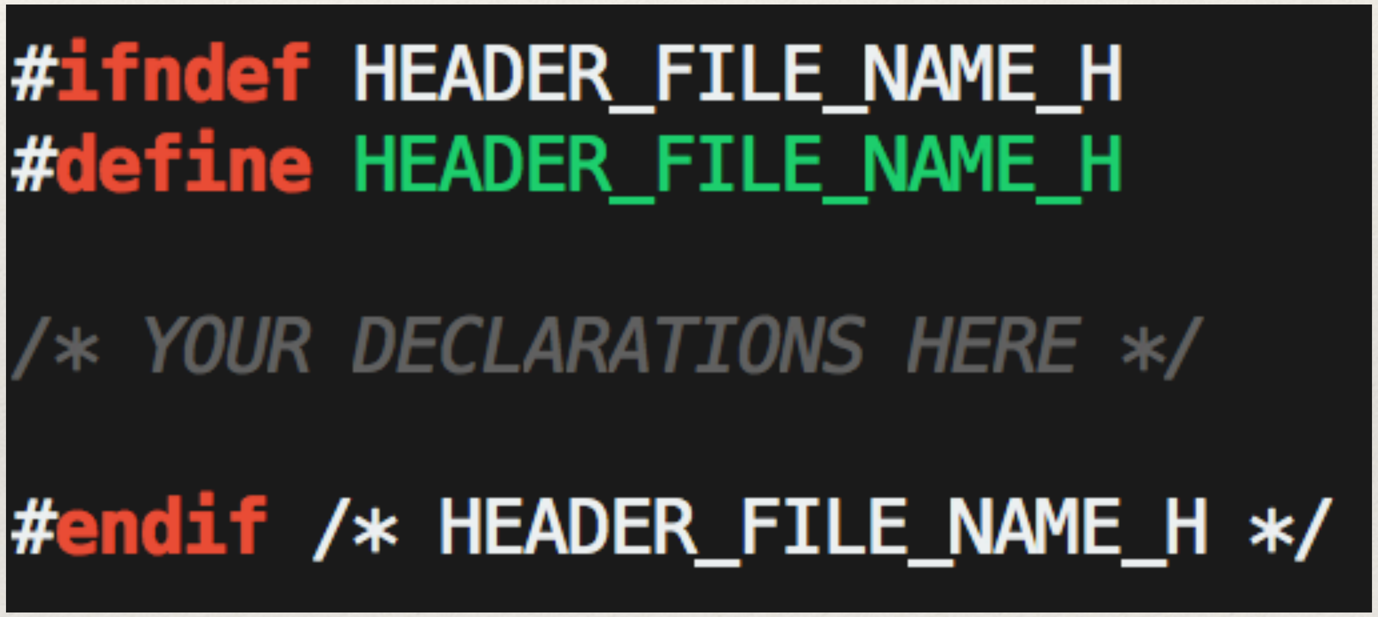
They will enter an infinite loop of constantly importing the other one and attempting to evaluate it.

To stop this from happening, we put something into our header files that forces them to only be evaluated once.

Safeguards in Header Files

In order to prevent a header file from being evaluated more than once, simply follow the model to the right:

This checks if the header file has been evaluated already, and only does so if it hasn't.



```
#ifndef HEADER_FILE_NAME_H
#define HEADER_FILE_NAME_H

/* YOUR DECLARATIONS HERE */

#endif /* HEADER_FILE_NAME_H */
```