# Header Files

## What's A Header File

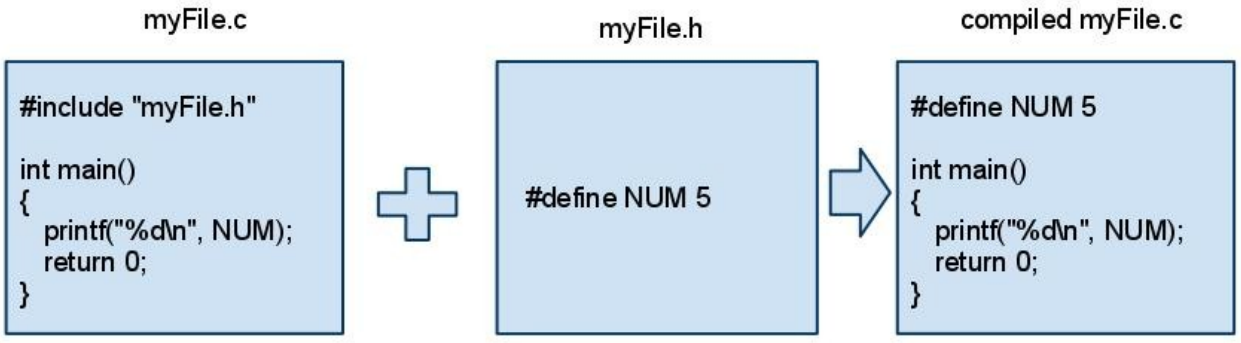
A header file is a place to store information that does not exclusively belong in a .c file. Imagine you have a struct that you want two different .c files to use. Instead of declaring the struct twice, you can just make a header file with that struct in it and include it in both files.

## What Happens to a Header File

When you #include your own header file make sure to surround it with quotes instead of angle brackets.

```
#include <myFile.h> //BAD
#include "myFile.h" //GOOD
```

When a file is #include'ed, the entire contents of the header file is just dumped into the top of the .c file.




## What to Put In a Header File

Since a header file gets dumped to the top of your file, things that used to go at the top of your file should be in your header file.

- structs
- #defines
- function prototypes
- other #includes

## What NOT to Put in a Header File

- variables (unless you are being very careful!)
- function definitions

## Structure of a Header File

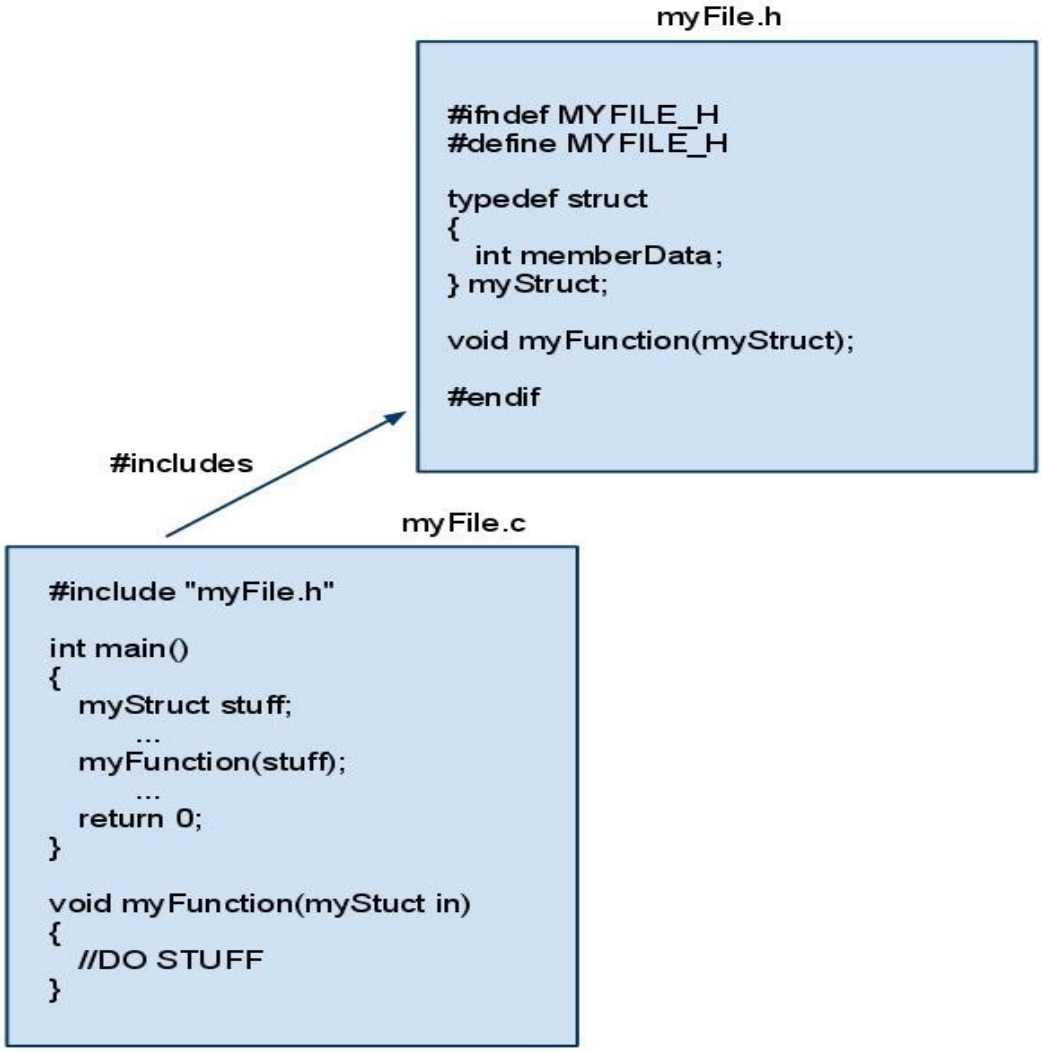
Header files are usually laid out like this:

```
//BEGIN File (this comment issn't actually in the file,
// its just to help you imagine a file)
#ifndef FILENAME_H //Replace FILENAME with your
                   // actual file name.
#define FILENAME_H //Ex: "lab12.h" => "LAB12_H"
```

```
        //Other #includes

        //#defines

        //structs

        //prototypes
#endif
//END FILE (once again, not actually in the file)
```

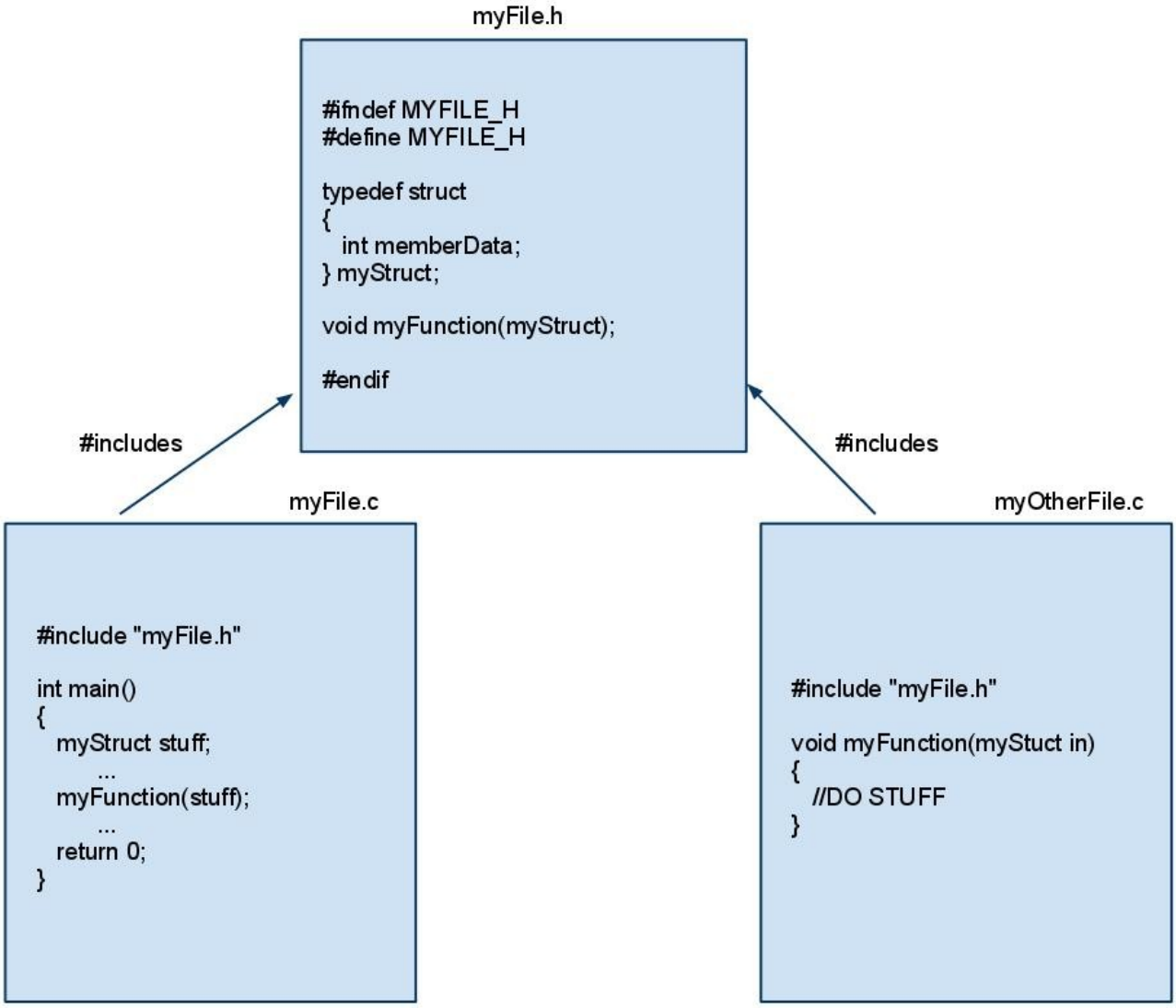
You probably noticed those #ifndef and #endif. These are to stop something called "double/multiple inclusion". That's when a file gets included more than once. It is not necessary, especially in simpler programs. If you don't understand, than don't worry too much about it.

## Single File Use




When you have a simple, single include program, you can compile it just like you would any other program: "gcc myFile.c". Notice how myFile.c uses stuff that is in the header file.

# Multiple File Use




Whenever you have multiple .c files, things get a bit more complicated. Notice how main only exists in one of the files. Look how both files needed the myStruct, so they share the header file. Also notice how myOtherFile.c defines the function “myFunction”, but myFile is still allowed to use it. This is because they both share the same header file. To compile this you will need to provide both the files on the command line: “gcc myFile.c myOtherFile.c”. The order of the files doesn't matter.