Separate Compilation of code

*If you want to read about love and marriage,
you've got to buy two separate books.
-- Alan King*

Definition vs Declaration

you can compile multiple files separately
which can be combined/linked.

multiple files

It is common to break a large problem up into smaller problems, and this can be helped by actually creating separate files for each code solution and combining them later (link time).

C++ is crazy about types though

How can I use a function that is defined in a different file?

- isn't C++ crazy about types? How does it know the types

You provide a function declaration!

declaration

A function ***declaration*** has no function body (no body, no code to execute).

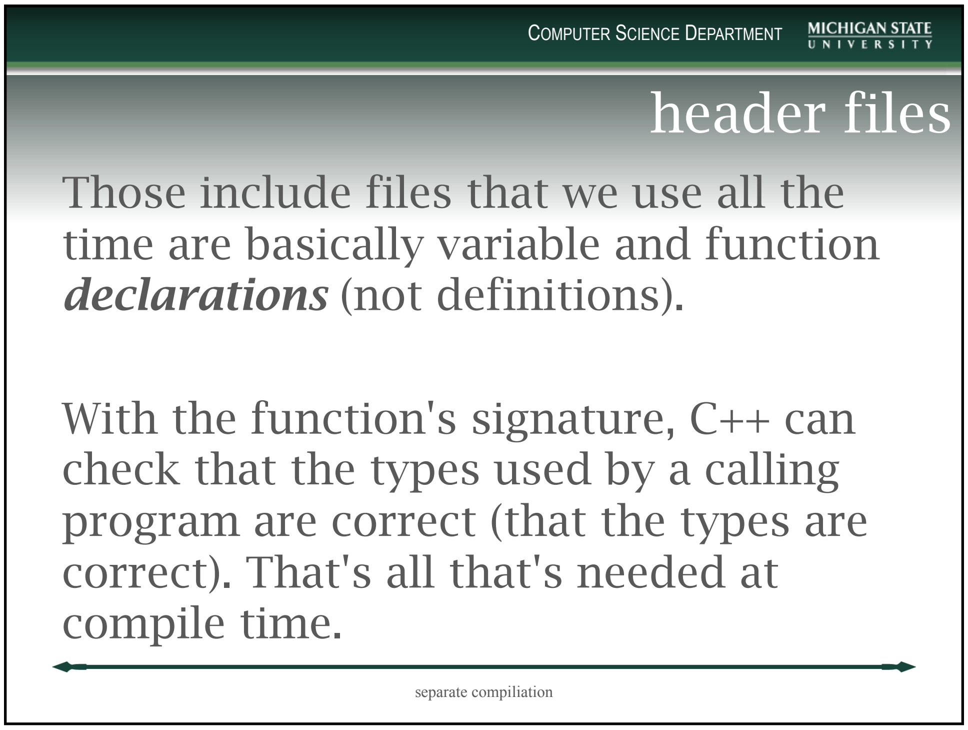
However, it does have ***all the types*** the function uses:

- knowing the types, the compiler can compile (check types) under the assumption that the function definition is provided later at link time

declaration == signature

A declaration registers the *signature* of a function, namely:

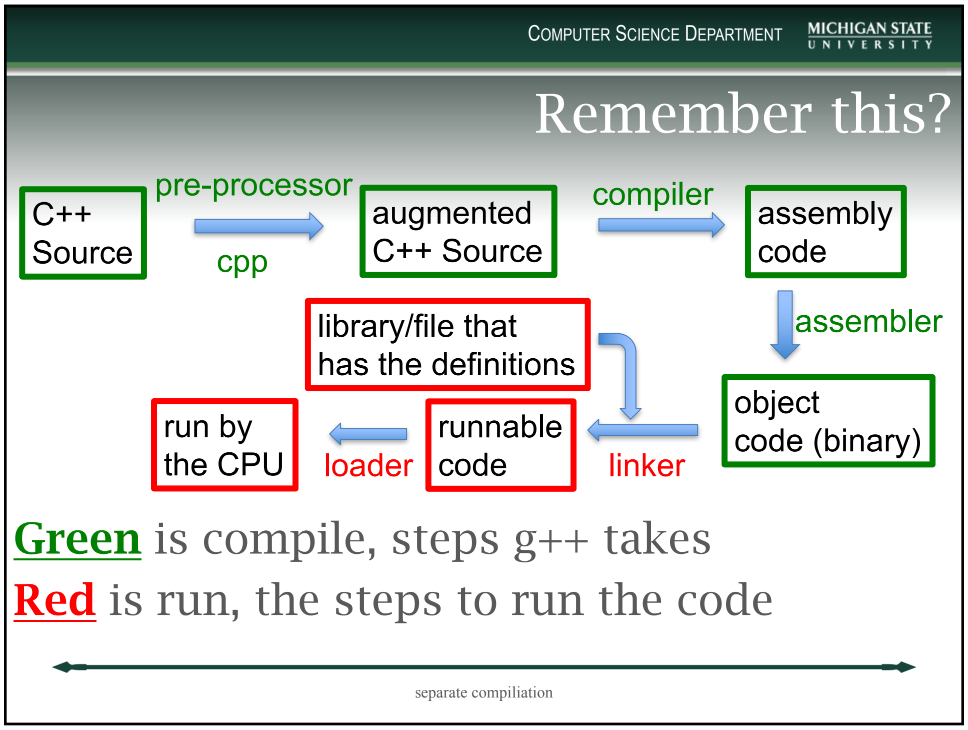
- its name
- its return type
- its parameter type(s)



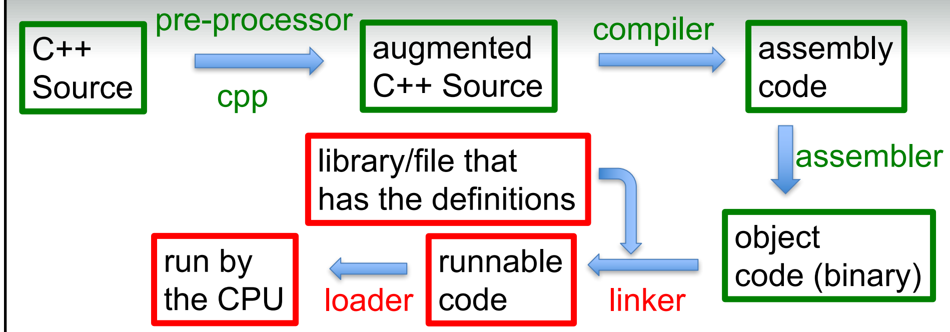
header files

Those include files that we use all the time are basically variable and function ***declarations*** (not definitions).

With the function's signature, C++ can check that the types used by a calling program are correct (that the types are correct). That's all that's needed at compile time.



Remember this?



Green is compile, steps g++ takes

Red is run, the steps to run the code

compile but don't link

We can instruct the compiler to compile but not link with `-c` flag

```
g++ -std=c++11 -Wall -g -c my_file.cpp
```

compile but don't link

produces a .o file

```
my_file.o
```

an object file to be linked. Not executable!

run the linker

Subsequently we can run the linker, on the .o files to produce an executable:

```
g++ file1.o file2.o -o outfile.exe
```

the `-o` allows us to name the executable (any name we like, I chose `.exe` here). Note one of those `.o` files must have a `main` function or you get a link error!

declaration:

```
void swap (long &, long&);
```

semi at the end of decl
param names not necessary.

definition

```
void swap(long &first, long &second){
   long temp = first;
   first = second;
   second = temp;
}
```

Characteristics

- Reusable
- Hide Implementation Details
- Ease Maintenance
- Permit Separate Compilation
- Support Independent Coding
- Simplify Testing

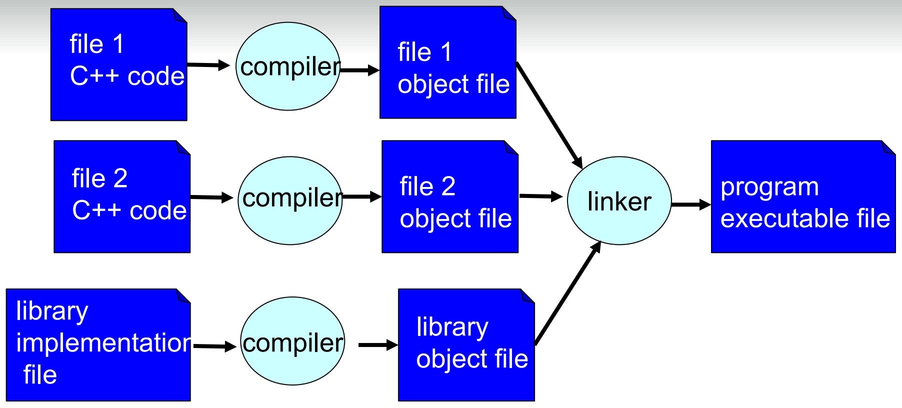
Libraries

- Libraries increase the power of a language because they allow commonly-used functions to be shared.
- We have used the Math library
- Today we will learn how to create our own

Library Components

- Header File
  - interface
  - declarations
  - inserted into using program using `#include`
- Implementation File
  - (function) definitions
  - must be separately compiled and then ***linked by the compiler***

Separate Compilation



Public/Private

- Public

  Items defined in the implementation file that are declared in the header file can be used by any program which includes the header.
- Private

  Items defined in the implementation file that are *not* declared in the header file *cannot* be used by any program, even if they include the header file.

Example 9.1

header files, separate compilation

header file

```
int get_coefficients( double &, double &, double & );
int roots( double, double, double, double &, double & );
```

you can `include` this file in your main.

it's enough to indicate the types

- notice no param names
  - not necessary, only the types
  - function name ***is*** required

Declaration

```
int get_coefficients( double &, double &, double & );
int roots( double, double, double, double &, double & );
```

Definition

```
int get_coefficients( double & A, double & B, double & C ){
    int status;

    cout << "Please enter the coefficients" << endl;
    cout << "(space separated A, B and C, or EOF to halt):"
         << endl;
    cin >> A >> B >> C;

    if (cin.eof())
   status = 0;
    else
   status = 1;

    return status;
}
```

different include

When you include header files from the C++ libraries or other standard tools, you use < >

```
#include <iostream>
```

When you include from your own local directory, you use " "

```
#include "myheader.hpp"
```

declaration before invocation

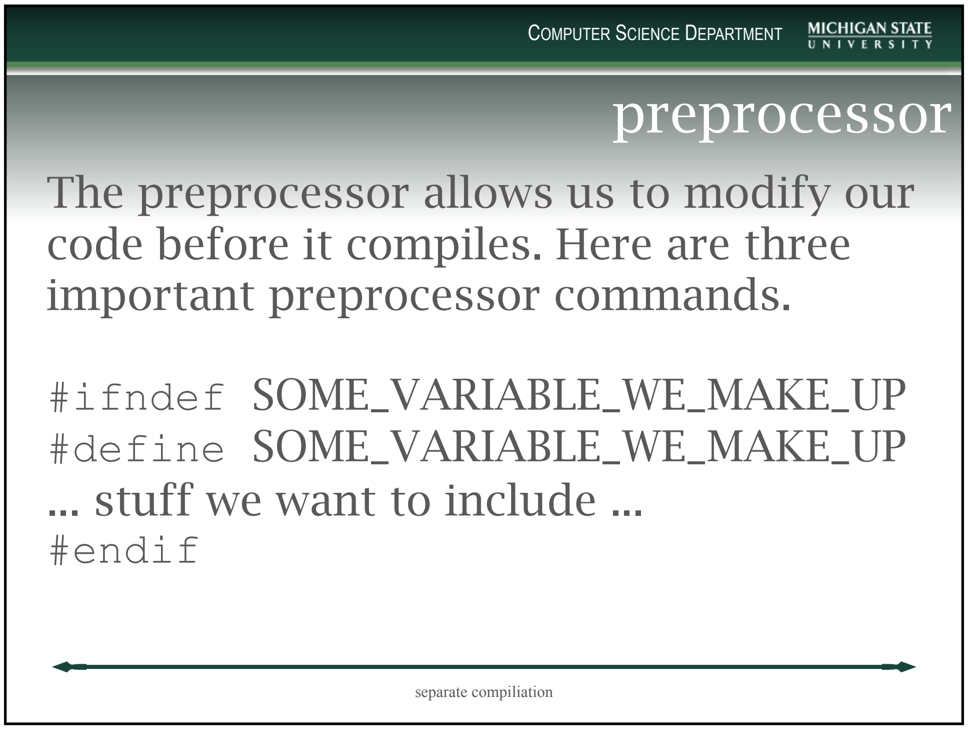
The definition (the body of the function) does not have to be in the main file.

However, the declaration (either in the file or through an include) has to occur **before** the invocation.

avoid multiple declarations

We can have the following problem. If we include a file more than once, then we are essentially declaring the same element (function, variable, whatever) multiple times. That is not allowed.

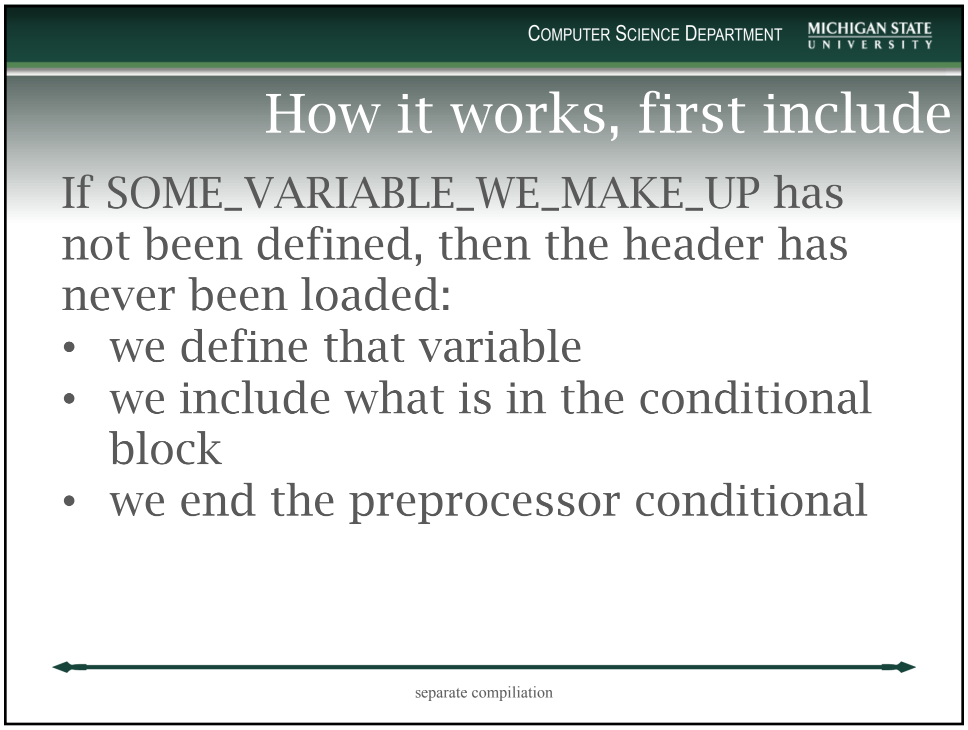
We do this all the time with provided headers, how do we avoid it?



preprocessor

The preprocessor allows us to modify our code before it compiles. Here are three important preprocessor commands.

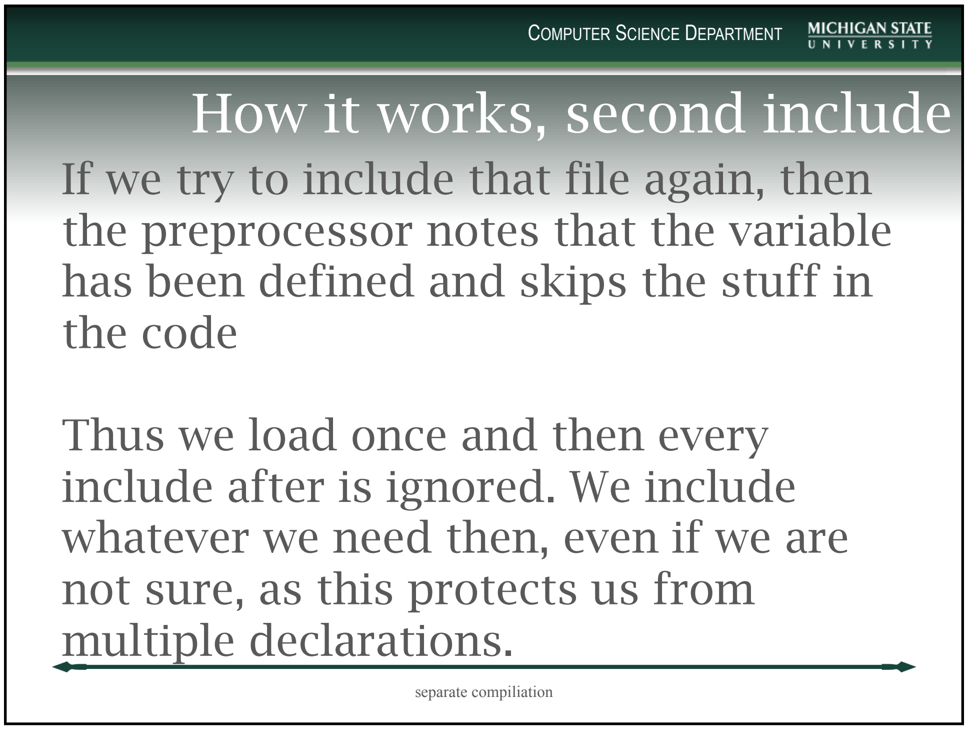
```
#ifndef SOME_VARIABLE_WE_MAKE_UP
#define SOME_VARIABLE_WE_MAKE_UP
… stuff we want to include …
#endif
```



How it works, first include

If SOME_VARIABLE_WE_MAKE_UP has not been defined, then the header has never been loaded:

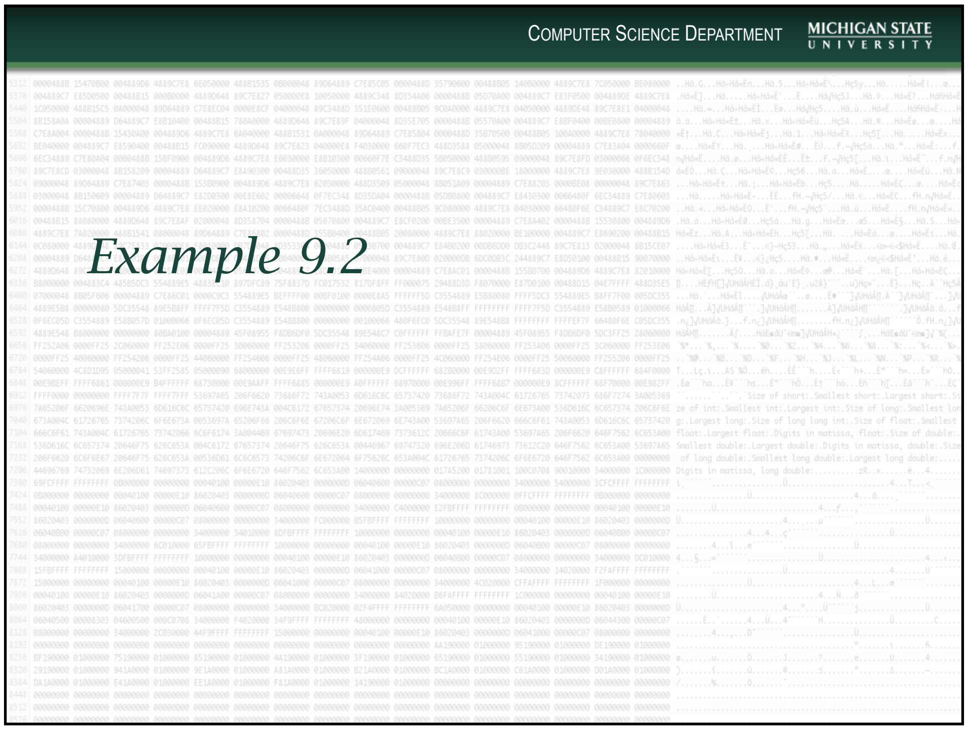
- we define that variable
- we include what is in the conditional block
- we end the preprocessor conditional



How it works, second include

If we try to include that file again, then the preprocessor notes that the variable has been defined and skips the stuff in the code

Thus we load once and then every include after is ignored. We include whatever we need then, even if we are not sure, as this protects us from multiple declarations.



Example 9.2

default values only in header

If you want to declare a default value for a parameter, you only do it in the header.

In the definition, you do not include the default. It is already there from the declaration.