Multiple File Compilation

Definition vs Declaration

- You can compile multiple files separately which can be combined / linked

Multiple Files

- It is common to break a large problem into smaller problems
- Can actually create separate files for each code solution
- Can combine them later (link time)

C++ is crazy about types

- How can I use a function that is defined in a different file?
  - How does it know the types?
- You provide a function declaration

Declaration

- A function **declaration** has no function body (no code to execute)
- 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 types

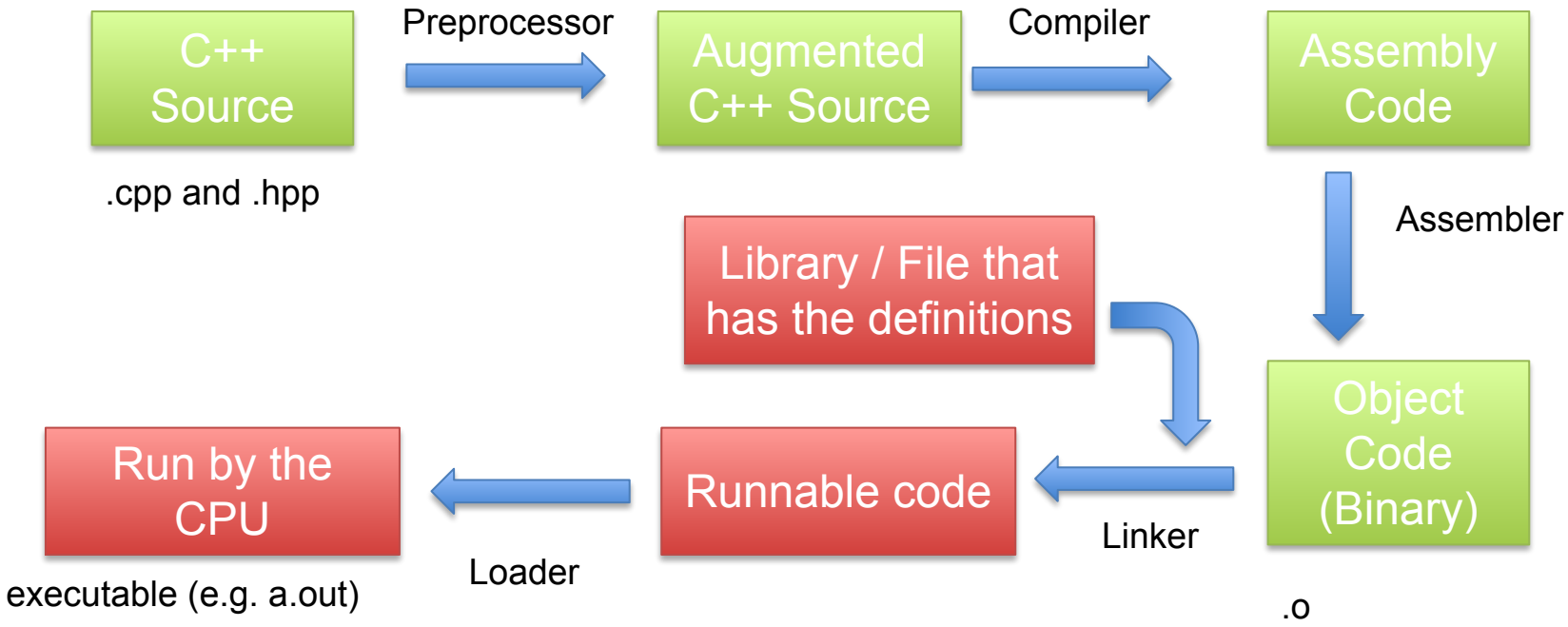
Which of the following is NOT part of a function's signature?

- Its name
- The names of its parameters
- The types of its parameters
- I don't know

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 is needed at compile time

Compile Execute Cycle



Green is compile-time, the steps g++ takes to compile the code

Red is runtime, the steps to run the code

Compile but don’t link

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

```
g++ -std=c++17 -Wall -g -c my_file.cpp
```

produces a `.o` file

`myfile.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).
- One of the `.o` files must have a main function or you get a link error!

Declaration vs Definition

- Declaration

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

Semicolon at the end of declaration
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

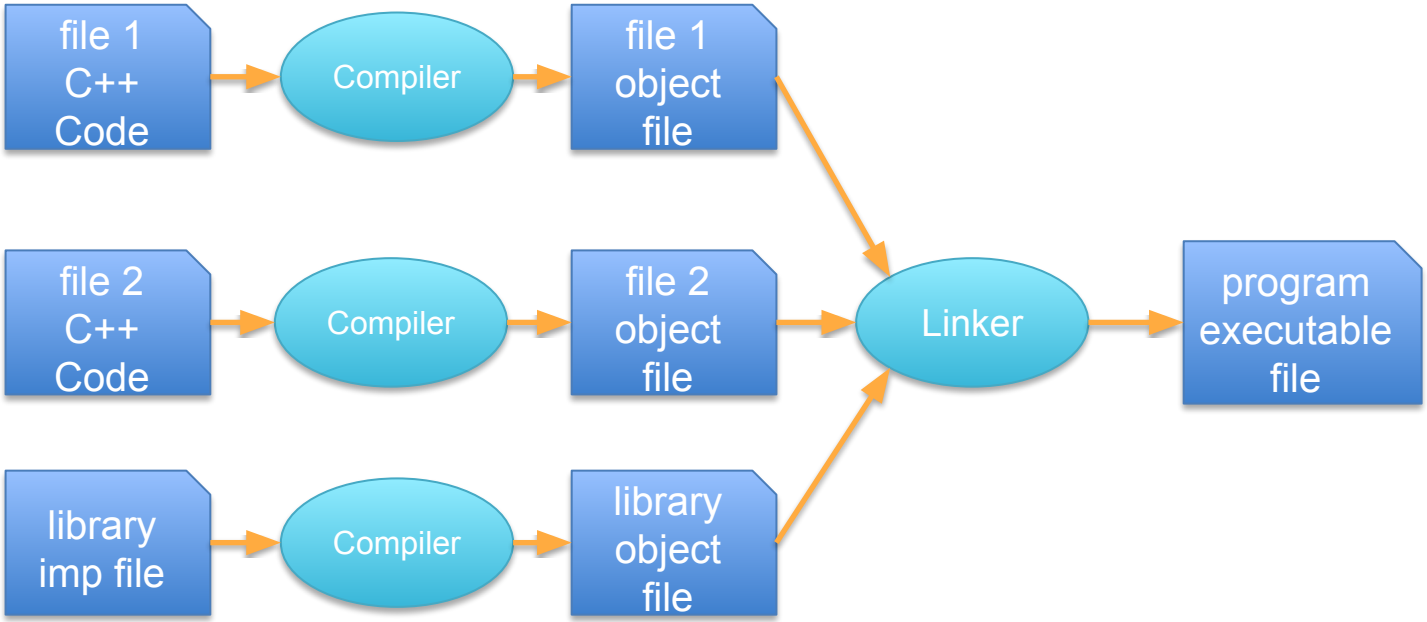
Libraries

- Libraries increases 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
  - Templates
  - Inserted into using program using `#include`
- Implementation file
  - (Function) Definitions
  - Classes (covered later)
  - 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 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

Multiple File Compilation (continued)

Which files needed to be compiled?

- All .cpp and .hpp files
- All .cpp files
- Just the .cpp file with the main function
- I don't know

Header file

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

- You can include this file in your main
- It is enough to indicate the types
  - Notice no param names
    - Not necessary (optional), only the types
    - Function name **is** required

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
- 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 than we are essentially declaring the same element (function, variable, whatever) multiple times.
- This 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

- It 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

What is the point of header guards
(like `#ifndef SOME_VARIABLE_WE_MAKE_UP`)?

- To make compilation faster
- To avoid multiple declarations
- To ensure headers have different namespaces
- I don't know

Example 9.2

#pragma once

- Preprocessor directive that fulfills the same purpose as include guards
- Some advantages
  - Don’t need to worry about unique guard variable names
  - May be faster
- Nonstandard but widely supported on newer compilers
  - gcc (g++) 3.4 or newer
  - MS Visual C++ (Visual Studio) 4.2 or newer
- Can combine `#pragma once` with include guards.

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.