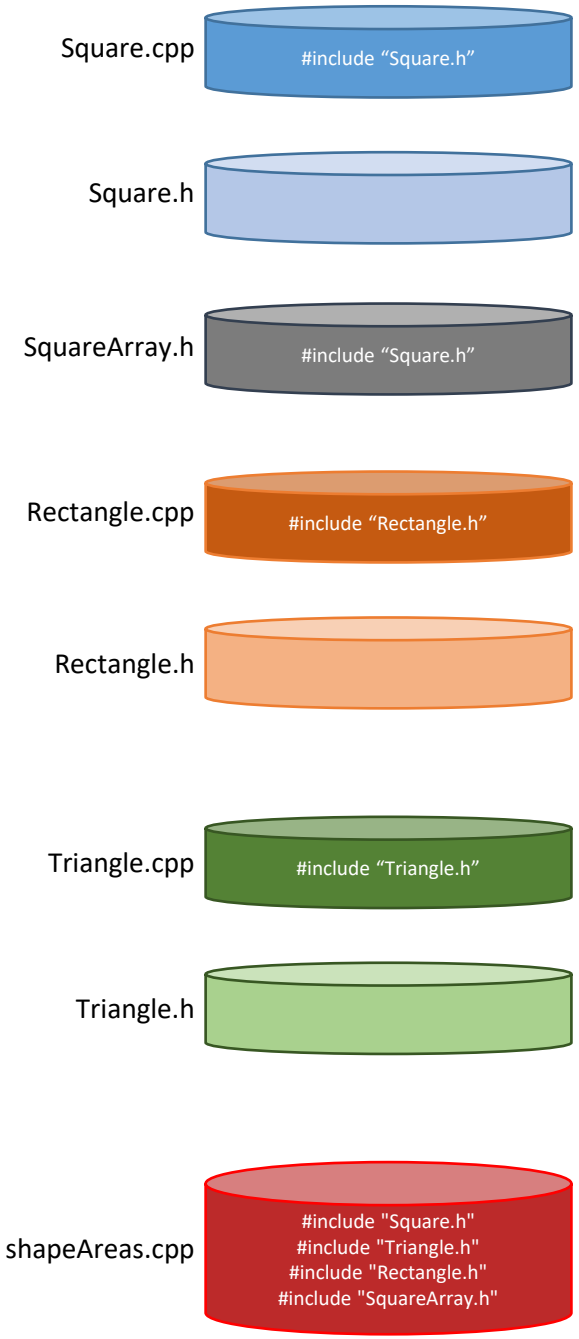


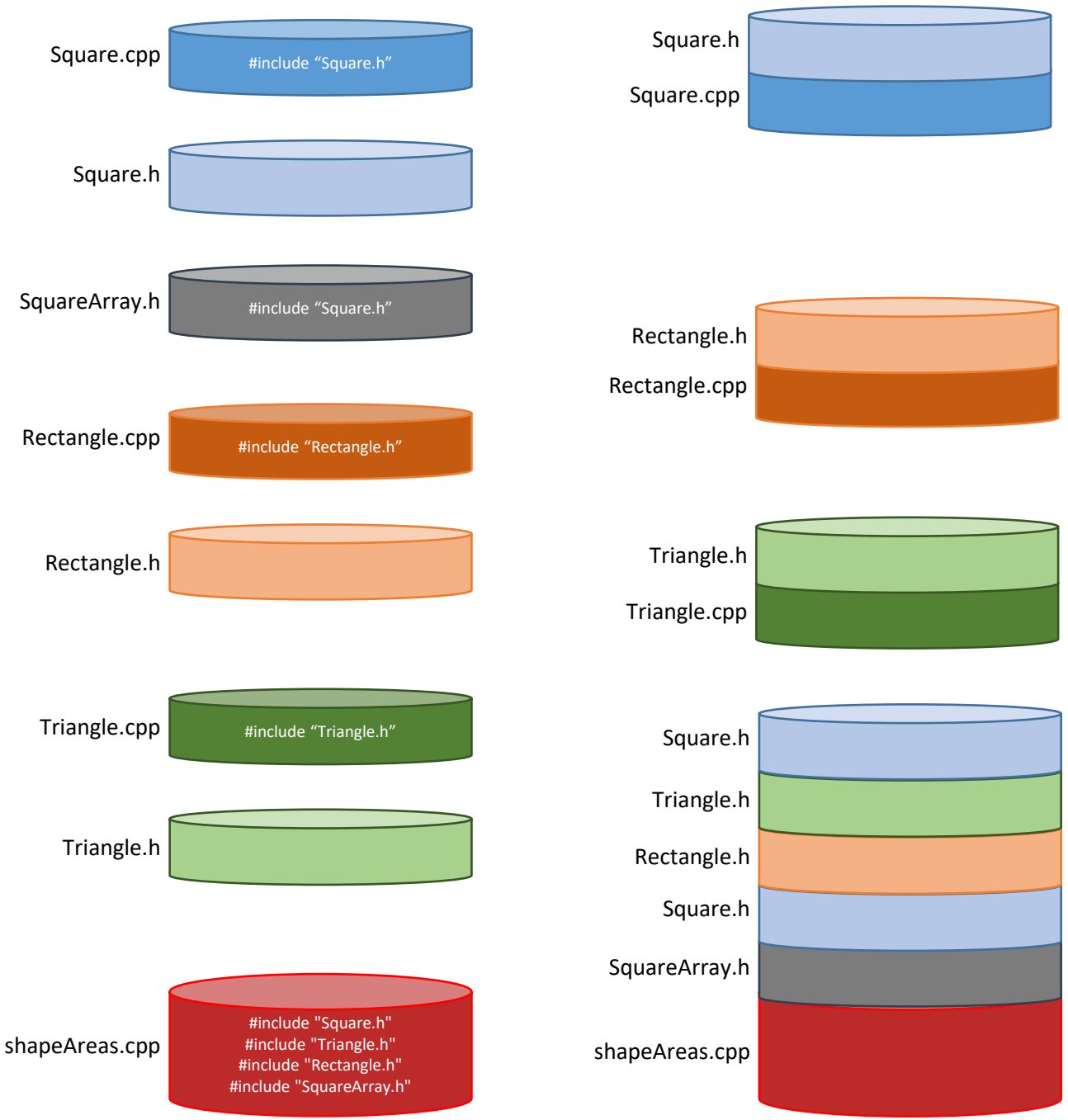
All the files involved in a simple project involving different shapes and their areas are shown here…

Class interfaces (class definitions with documented prototypes) go in .h files named the same as the class

Class implementations (the implement of the class member functions, bound to the class using scope resolution) go in .cpp files named the same as the class

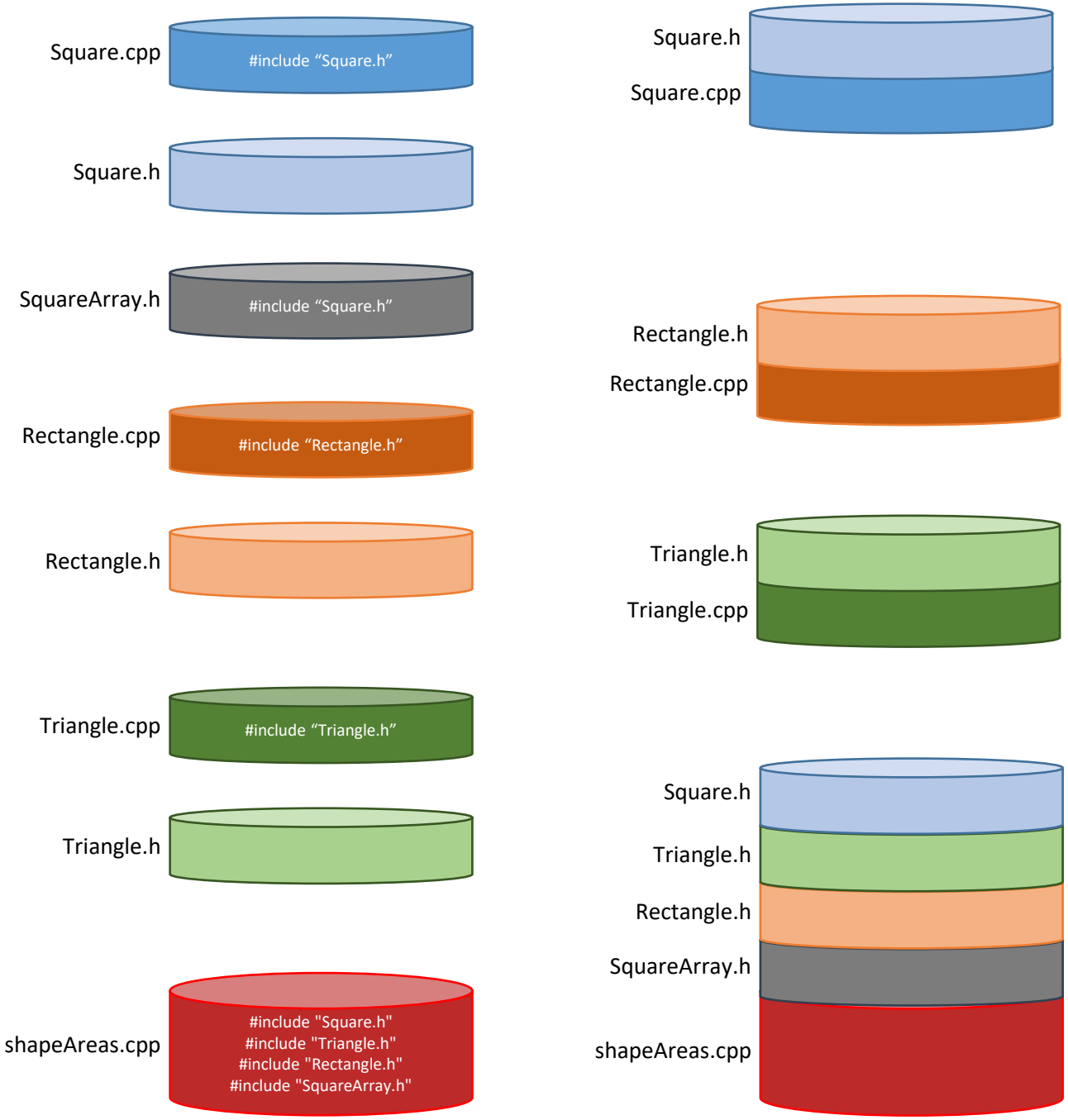
The main function goes in its own .cpp source file (shapeAreas.cpp in this example).

If you had other global functions (i.e. non-member functions):
- each (documented) global function prototype would go in its own .h file named the same as the function.
- each global function implementation would go in its own .cpp file named the same as the function.



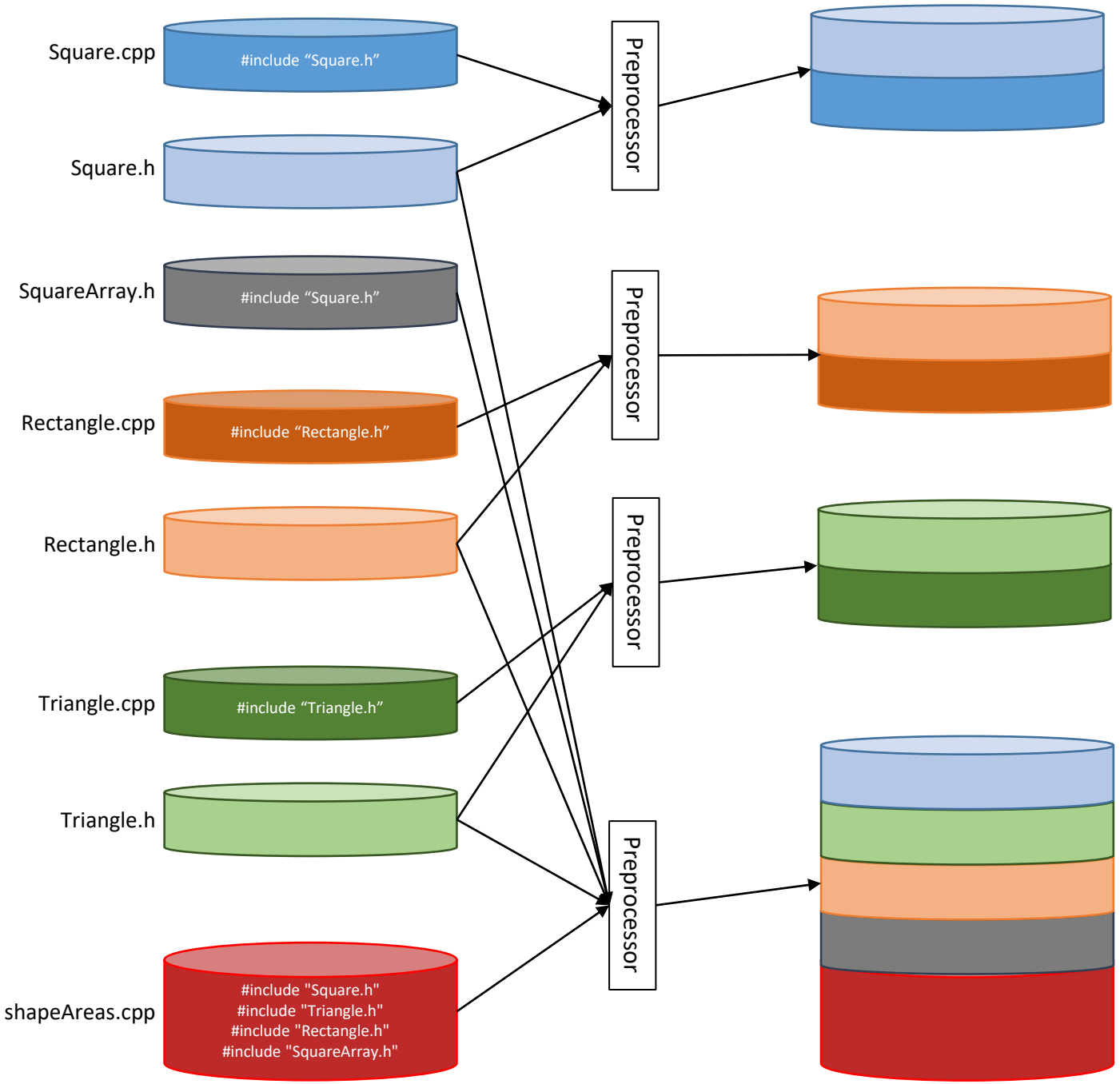
Example of preprocessor outputs, **without** #include guards

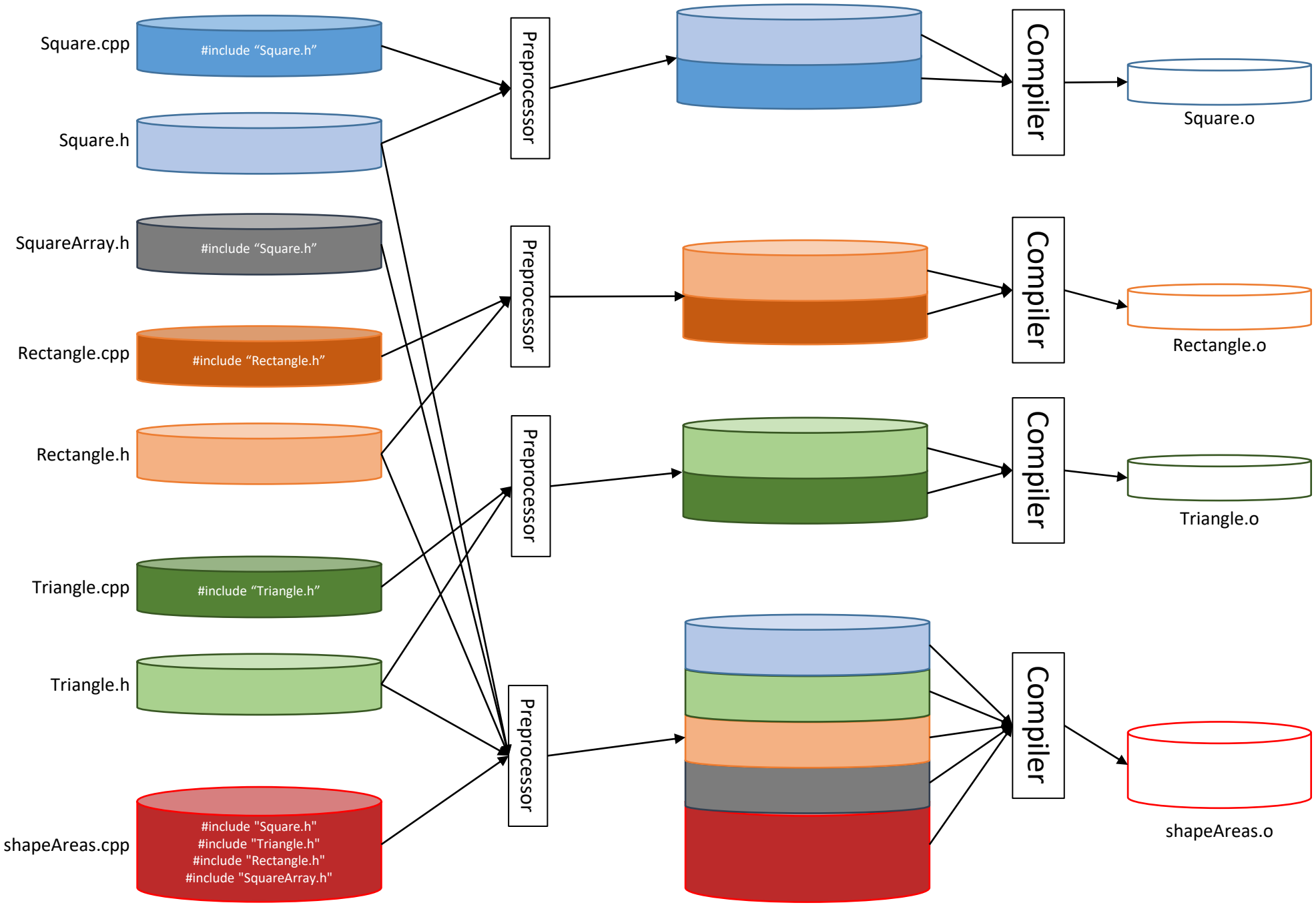
Use of #include guards prevents this header file from being “pasted” in multiple times, as shown on the next slide

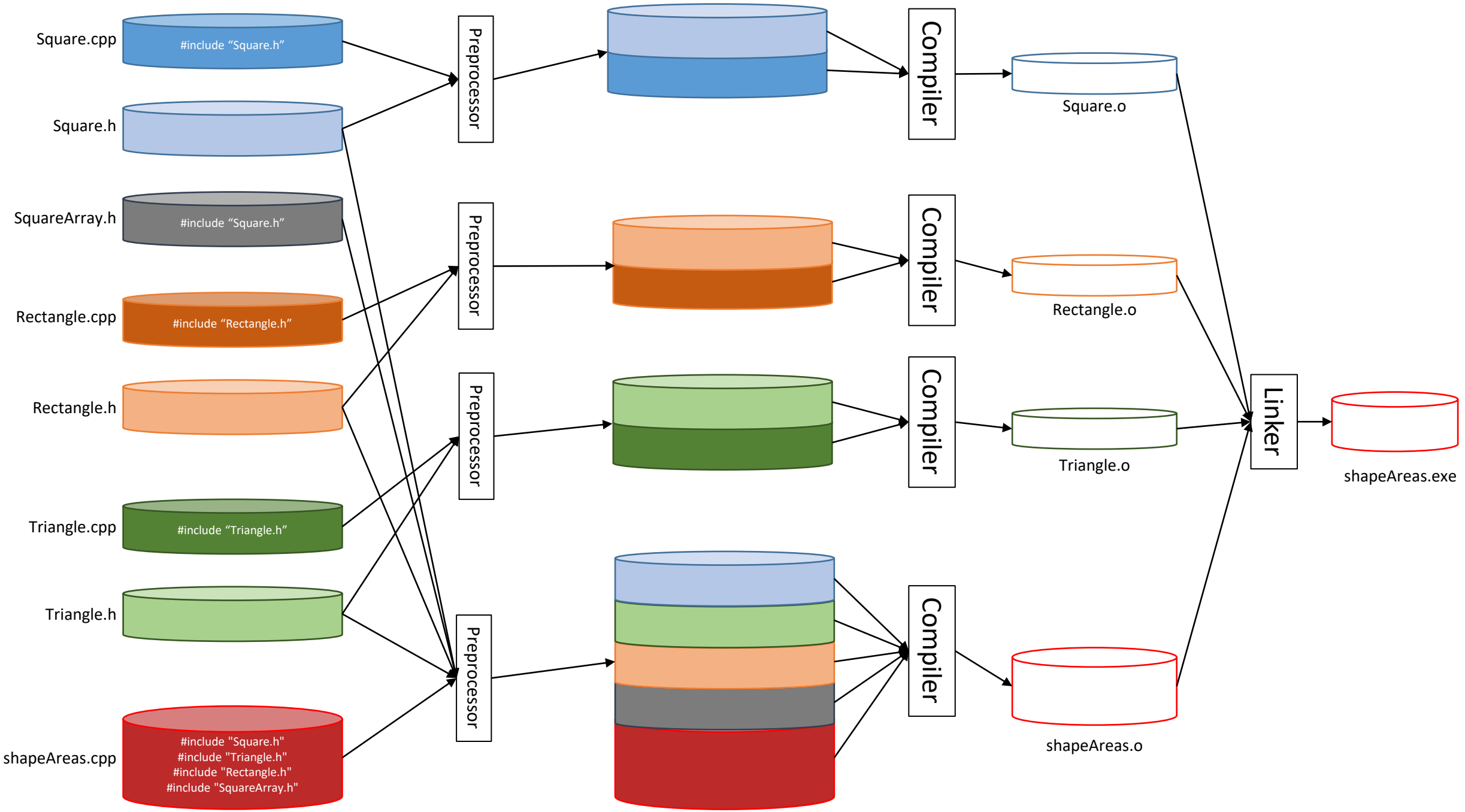


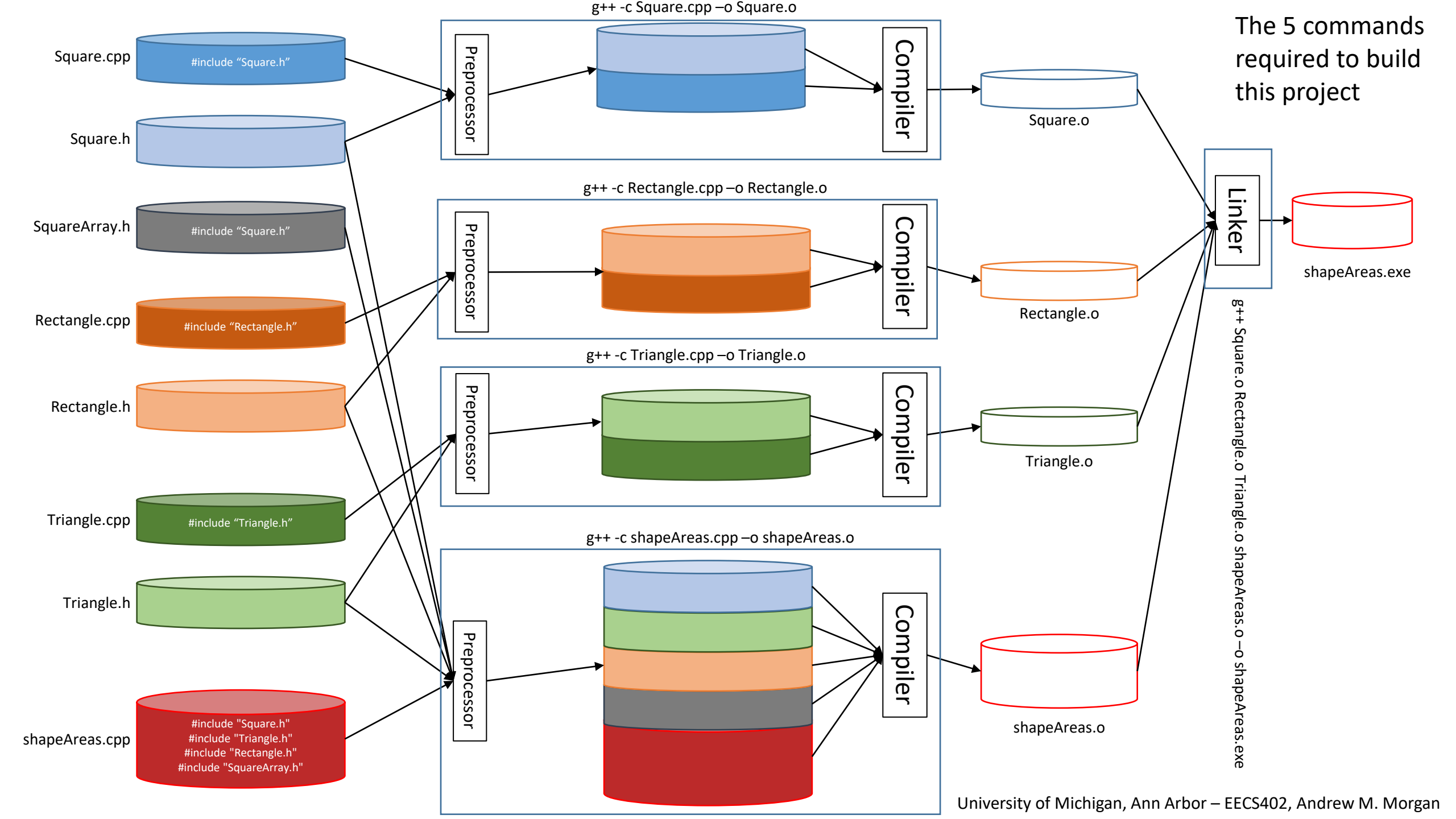
Example of preprocessor outputs, **with** #include guards

Everything is now “pasted” in exactly once!


Square.cpp
#include “Square.h”
Square.h
SquareArray.h
#include “Square.h”
Rectangle.cpp
#include “Rectangle.h”
Rectangle.h
Triangle.cpp
#include “Triangle.h”
Triangle.h
shapeAreas.cpp
#include "Square.h"
#include "Triangle.h"
#include "Rectangle.h"
#include "SquareArray.h"
Preprocessor
Preprocessor
Preprocessor
Preprocessor


Square.cpp
#include “Square.h”
Square.h
SquareArray.h
#include “Square.h”
Rectangle.cpp
#include “Rectangle.h”
Rectangle.h
Triangle.cpp
#include “Triangle.h”
Triangle.h
shapeAreas.cpp
#include "Square.h"
#include "Triangle.h"
#include "Rectangle.h"
#include "SquareArray.h"
Preprocessor
Preprocessor
Preprocessor
Preprocessor
Compiler
Compiler
Compiler
Compiler
Square.o
Rectangle.o
Triangle.o
shapeAreas.o

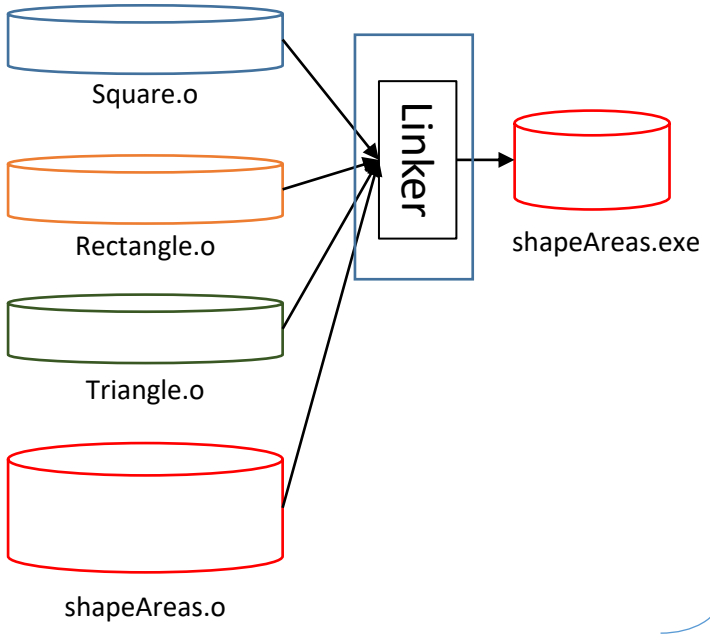

Square.cpp
#include “Square.h”
Square.h
SquareArray.h
#include “Square.h”
Rectangle.cpp
#include “Rectangle.h”
Rectangle.h
Triangle.cpp
#include “Triangle.h”
Triangle.h
shapeAreas.cpp
#include "Square.h"
#include "Triangle.h"
#include "Rectangle.h"
#include "SquareArray.h"
Preprocessor
Preprocessor
Preprocessor
Preprocessor
Compiler
Compiler
Compiler
Compiler
Square.o
Rectangle.o
Triangle.o
shapeAreas.o
Linker
shapeAreas.exe


The 5 commands required to build this project
Square.cpp
#include “Square.h”
Square.h
SquareArray.h
#include “Square.h”
Rectangle.cpp
#include “Rectangle.h”
Rectangle.h
Triangle.cpp
#include “Triangle.h”
Triangle.h
shapeAreas.cpp
#include "Square.h"
#include "Triangle.h"
#include "Rectangle.h"
#include "SquareArray.h"
g++ -c Square.cpp –o Square.o
Preprocessor
Compiler
Square.o
g++ -c Rectangle.cpp –o Rectangle.o
Preprocessor
Compiler
Rectangle.o
g++ -c Triangle.cpp –o Triangle.o
Preprocessor
Compiler
Triangle.o
g++ -c shapeAreas.cpp –o shapeAreas.o
Preprocessor
Compiler
shapeAreas.o
Linker
shapeAreas.exe
g++ Square.o Rectangle.o Triangle.o shapeAreas.o –o shapeAreas.exe

Determining Dependencies:

Look at the commands and the arrows leading into the box

Examples:

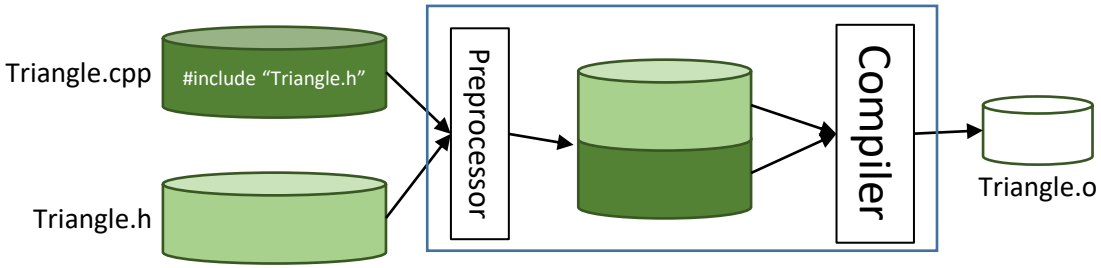


In order to build shapeAreas.exe (the "target"), the linker needs to be executed, and it depends on Square.o, Rectangle.o, Triangle.o, and shapeAreas.o.

Note: for this case, the .o files indicated are all of the dependencies – the linker does not depend on any of the .cpp or .h files to do its job!

In the Makefile, this command would look like this:

```
shapeAreas.exe: Square.o Rectangle.o Triangle.o shapeAreas.o
        g++ Square.o Rectangle.o Triangle.o shapeAreas.o –o shapeAreas.exe
```



In order to build Triangle.o (the "target"), "g++ -c" needs to be executed, and it depends on Triangle.cpp and Triangle.h.

In the Makefile, this command would look like this:

```
Triangle.o: Triangle.cpp Triangle.h
        g++ -c Triangle.cpp –o Triangle.o
```

The Makefile for the project

Targets before the colons

Dependencies after the colons

Commands to execute after the target line following a *tab* character

“all” is a special target – specify it first, and have it depend on the “main” thing(s) you want to build

“clean” is a special target – usually specific last, and simply removes all the “by-products” of the build process so you can “start fresh”

```
all: shapeAreas.exe

shapeAreas.o: shapeAreas.cpp Triangle.h Square.h Rectangle.h SquareArray.h
	g++ -Wall -c shapeAreas.cpp -o shapeAreas.o

Triangle.o: Triangle.cpp Triangle.h
	g++ -Wall -c Triangle.cpp -o Triangle.o

Square.o: Square.cpp Square.h
	g++ -Wall -c Square.cpp -o Square.o

Rectangle.o: Rectangle.cpp Rectangle.h
	g++ -Wall -c Rectangle.cpp -o Rectangle.o

shapeAreas.exe: Triangle.o Square.o Rectangle.o shapeAreas.o
	g++ Triangle.o Square.o Rectangle.o shapeAreas.o -o shapeAreas.exe

clean:
	rm -f *.o *.exe
```