Object-Oriented Programming

Separate Compilation

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

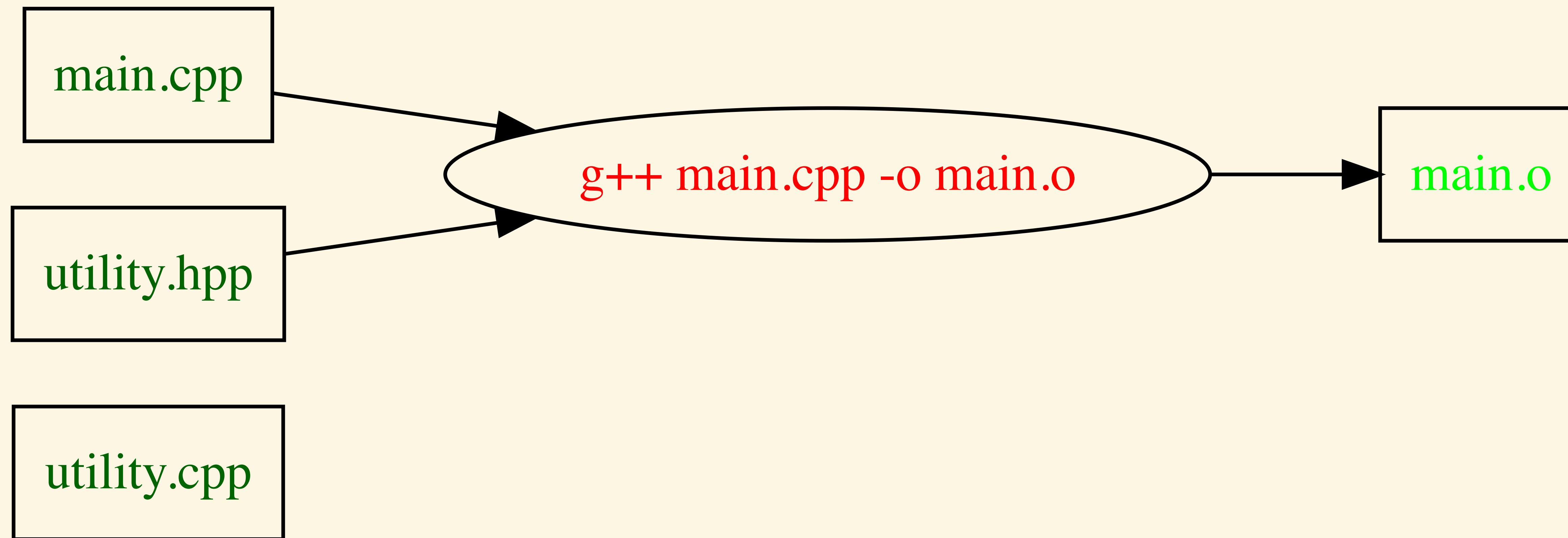
Build: Start

main.cpp

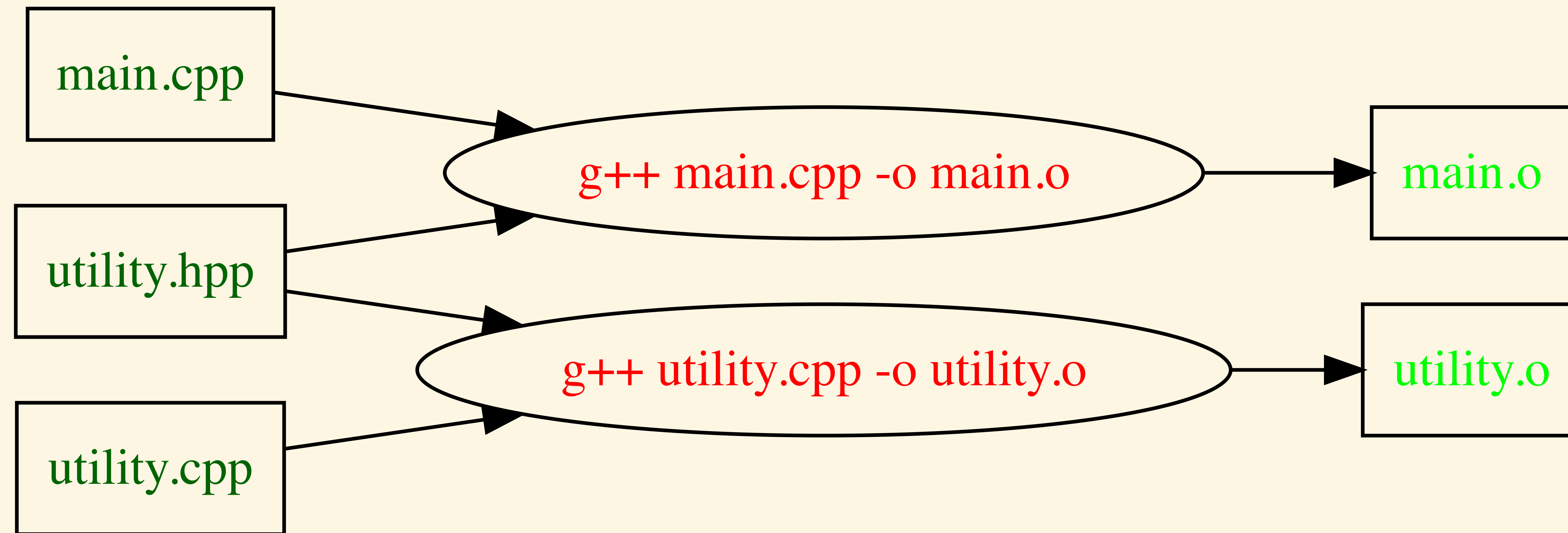
utility.hpp

utility.cpp

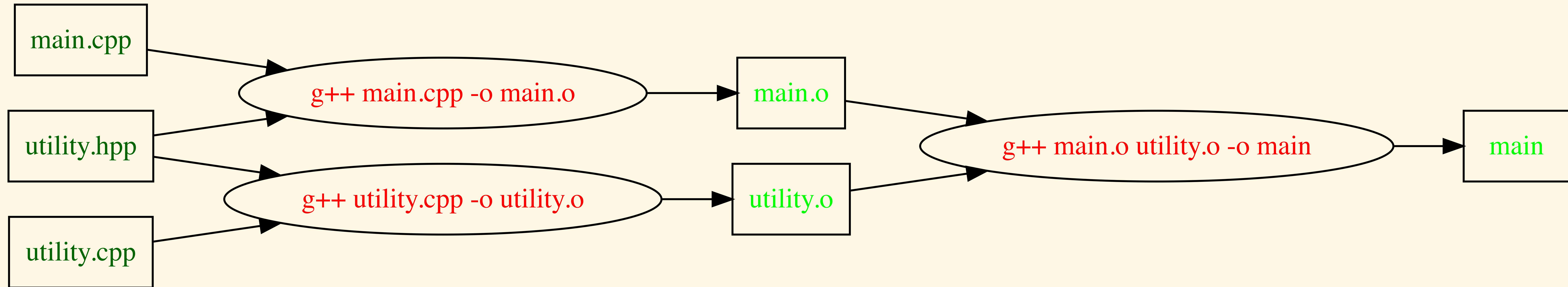
Build: Compile main.cpp



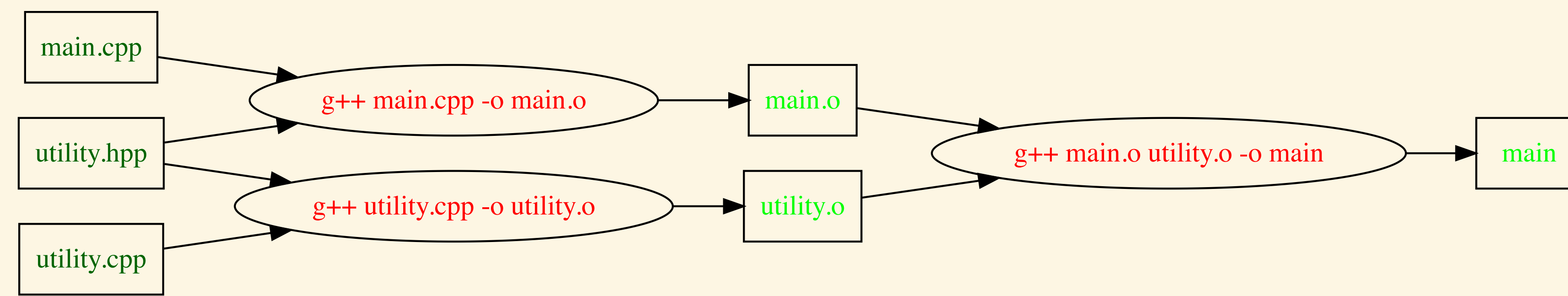
Build: Compile utility.cpp



Build: Link



Separate Compilation



- Real programs are composed of multiple source-code files
- Source-code files, i.e., `*.cpp`, are compiled separately into *object files*
- The object files are *linked* together using the *linker*
- For each application, one file is the *main program* and contains the *main function*
- Separating functions and classes into separate files increases *cohesion* (good) but also increases *coupling* (can be bad)

File Structure

Type	Purpose	Extension	Example
Include File	Function declarations	*.hpp*	*Aggregate.hpp*
Implementation File	Function definitions	*.cpp*	*Aggregate.cpp*

Include File

```
/*
    Functions.hpp

    Include file for ...
*/

#ifndef INCLUDED_FUNCTIONS_HPP
#define INCLUDED_FUNCTIONS_HPP

// Insert: Include files necessary for compiling
//         function declarations

// Insert: Function declarations

#endif
```

- Required header comment, as in any other source file
- Required *include guard* to prevent the compiler from seeing these contents more than once
- Insert **only** the include of any files needed for the *function declarations*
- Insert the *function declarations* (*function prototypes*)
- The code that uses the include file **must** compile no matter where it is included or which includes precede it

Include Guard Macro Names

```
/*
    Functions.hpp

    Include file for ...
*/

#ifndef INCLUDED_FUNCTIONS_HPP
#define INCLUDED_FUNCTIONS_HPP

// Insert: Include files necessary for compiling
//         function declarations

// Insert: Function declarations

#endif
```

- Generate the name based on the name of the file: All uppercase, and replace dot `'.'` with an underscore `'_'`
- May include a standard prefix (separated by an underscore). I use the prefix `INCLUDED_` (see [Lakos])
- If there are underscores in the name, use them. Do not add underscores for anything else

Implementation File

```cpp
/*
    Functions.cpp

    Implementation file for ...
*/

#include "Functions.hpp"

// Insert: Include files necessary for compiling
//         function definitions

// Insert: Function definitions
```

- Required header comment, as in any other source file
- Include the related include file **first**, then when you compile the implementation file, you check that the include file is complete.
- Insert any included files needed for the *implementation*
- An include file is needed in the implementation file if it is used in the implementation of the code, i.e., the *function definition*
- **Do not include .cpp files**

- In th
a 0 o

Roles

Type	Purpose	Extension	Example	Roles
Include File	Function declarations	*.hpp*	*Aggregate.hpp*	*interface concerns*
Implementation File	Function definitions	*.cpp*	*Aggregate.cpp*	*implementation concerns*

Design Guideline

Hide as many concerns as possible in the implementation file

- The **only** include files in the include file should be those needed for the *interface*
- This means the include files needed for code that *calls* the functions provide by the *interface*
- Include files used only in the *implementation* belong in the implementation file

Indirect Includes

```cpp
/*
    Label.hpp

    Include file for label functions.
*/

#ifndef INCLUDED_LABEL_HPP
#define INCLUDED_LABEL_HPP

#include <string>

// output the label
void outputLabel(std::string label);

#endif
```

```cpp
/*
    ClassName.cpp

    Output the class name.
*/

#include "Label.hpp"

int main() {

    std::string className = "OOP";

    outputLabel(className);

    return 0;
}
```

```cpp
/*
    Label.hpp

    Include file for label functions.
*/

#ifndef INCLUDED_LABEL_HPP
#define INCLUDED_LABEL_HPP

#include <string_view>

// output the label
void outputLabel(std::string_view label);

#endif
```

Direct Includes

```
/*
    Label.hpp

    Include file for label functions.
*/

#ifndef INCLUDED_LABEL_HPP
#define INCLUDED_LABEL_HPP

#include <string>

// output the label
void outputLabel(std::string label);

#endif
```

```
/*
    ClassName.cpp

    Output the class name.
*/

#include "Label.hpp"
#include <string>

int main() {

    std::string className = "OOP";

    outputLabel(className);

    return 0;
}
```

```
/*
    Label.hpp

    Include file for label functions.
*/

#ifndef INCLUDED_LABEL_HPP
#define INCLUDED_LABEL_HPP

#include <string_view>

// output the label
void outputLabel(std::string_view label);

#endif
```

Self-Sufficiency

```
/*
    main.cpp

    ...
*/

#include "Label.hpp"

int main() {

    return 0;
}
```

- An include file needs to be self-sufficient
- It should not depend on previous includes or the order of includes
- This applies to all include files that we write

IWYU

```
/*
    ClassName.cpp

    Output the class name.
*/

#include "Label.hpp"
#include <string>

int main() {

    std::string className = "OOP";

    outputLabel(className);

    return 0;
}
```

- *Include **W**hat **Y**ou **U**se*
- In every .cpp or .hpp, include any needed include files
- **Do not depend on indirect includes**
- This allows implementers of the include files to change their dependencies

Exposing Implementation Concerns

```cpp
/*
    Label.hpp

    Include file for label functions.
*/

#ifndef INCLUDED_LABEL_HPP
#define INCLUDED_LABEL_HPP

#include "pdfgen.h"
#include <string_view>

// output the label
void outputLabel(std::string_view label);

#endif
```

```cpp
/*
    Label.cpp

    Implementation file for label functions.
*/

#include "Label.hpp"

// output the label
void outputLabel(std::string_view label) {

    struct pdf_info info = {
        .creator = "My software",
        .producer = "My software",
        .title = "My document",
        .author = "My name",
        .subject = "My subject",
        .date = "Today"
    };
    struct pdf_doc *pdf = pdf_create(PDF_A4_WIDTH, PDF_A4_HEIGHT, &info);
    pdf_set_font(pdf, "Times-Roman");
    pdf_append_page(pdf);
    pdf_add_text(pdf, NULL, label.data(), 12, 50, 20, PDF_BLACK);
    pdf_save(pdf, "output.pdf");
    pdf_destroy(pdf);
}
```

Hiding Implementation Concerns

```cpp
/*
    Label.hpp

    Include file for label functions.
*/

#ifndef INCLUDED_LABEL_HPP
#define INCLUDED_LABEL_HPP

#include <string_view>

// output the label
void outputLabel(std::string_view label);

#endif
```

```cpp
/*
    Label.cpp

    Implementation file for label functions.
*/

#include "Label.hpp"
#include "pdfgen.h"

// output the label
void outputLabel(std::string_view label) {

    struct pdf_info info = {
        .creator = "My software",
        .producer = "My software",
        .title = "My document",
        .author = "My name",
        .subject = "My subject",
        .date = "Today"
    };
    struct pdf_doc *pdf = pdf_create(PDF_A4_WIDTH, PDF_A4_HEIGHT, &info);
    pdf_set_font(pdf, "Times-Roman");
    pdf_append_page(pdf);
    pdf_add_text(pdf, NULL, label.data(), 12, 50, 20, PDF_BLACK);
    pdf_save(pdf, "output.pdf");
    pdf_destroy(pdf);
}
```

Summary

- The include file, e.g., Aggregate.hpp, is for the **callers** of a function

- Want to make it as easy as possible for the callers, so only include what is needed for the **caller** of the function

- Every include file is a *dependency*. The fewer dependencies, the better.

- Keep as many details hidden in the implementation file, e.g., Aggregate.cpp