CSCI 104 Lab 3:

Makefiles

Lab 3: Makefiles

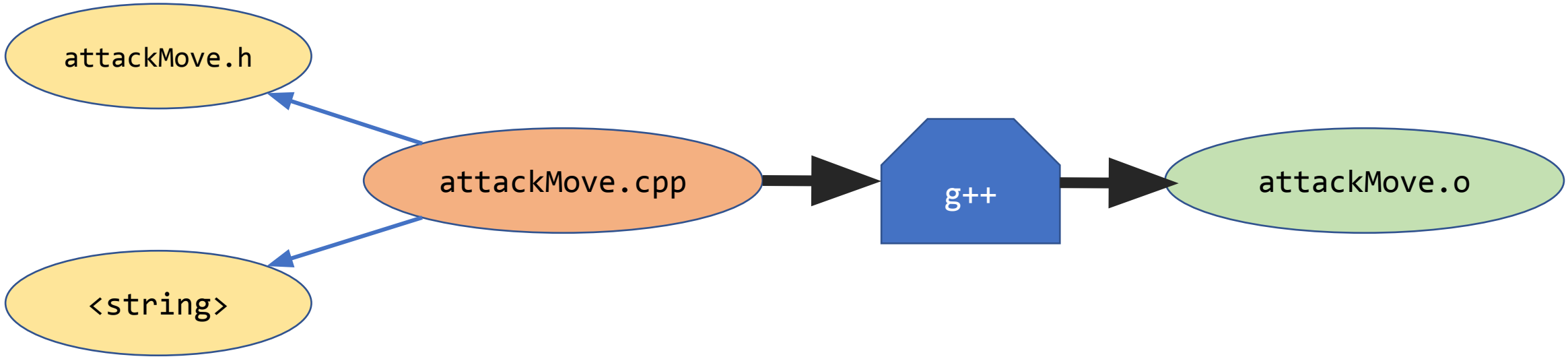
- Make is an extremely useful tool to make building your code less tedious
- Can be used to automatically build one or more programs with a single command
- *Incremental builds* – Make only rebuilds the parts of your code that have changed
- In order to tell Make what exactly it needs to do, you need to write a Makefile
- This contains a list of each file to compile, what it depends on, and how to build it.

Building C++ Code

- Two basic steps in building C++ programs: *compilation* and *linking*.
- Compilation: Functions in a source file are turned into machine code.
  - Machine code stored in an *object file* (.o)
  - Errors in the *source code* are detected at this time.
- Linking: Files containing object code are merged together and a single program is created.
  - This generates an executable you can run
  - Errors in the *complete program* are detected at this time.
  - In particular, missing functions will be detected with an “undefined symbol” error.

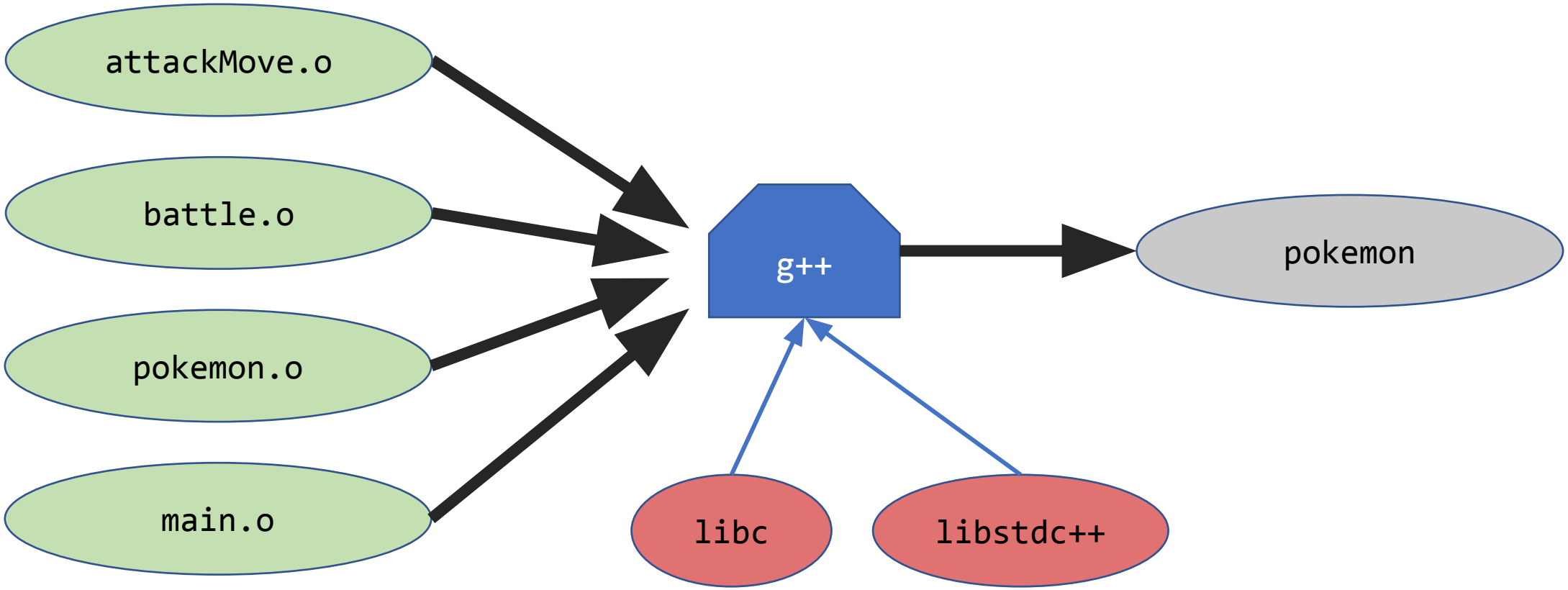
Compile Step

The compiler processes the entire source code for a file (including any headers you include) and generates the equivalent machine code.



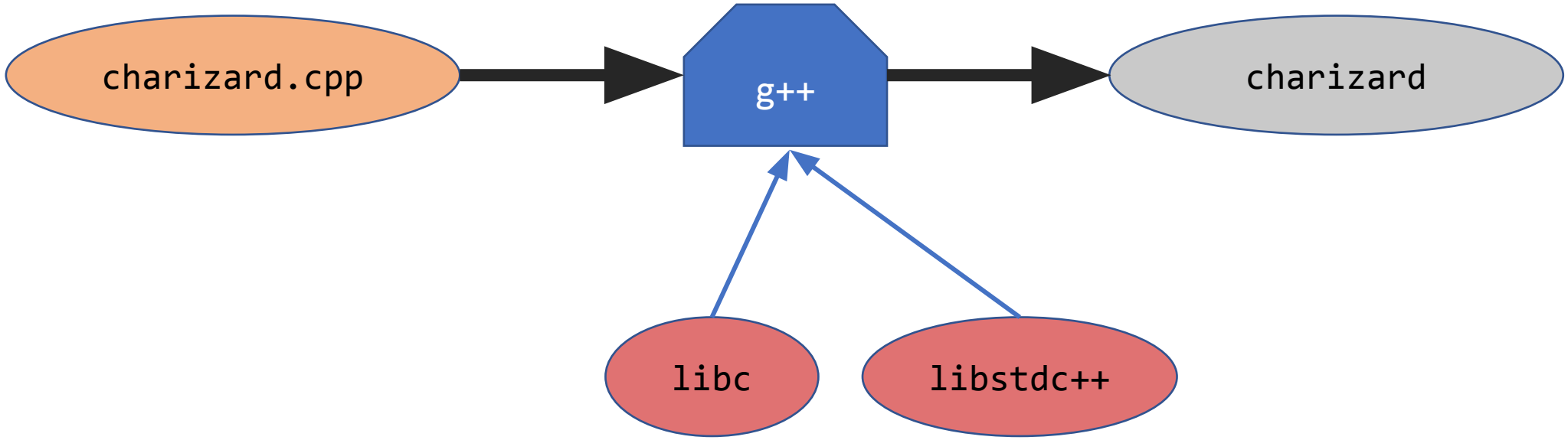
Link Step

The compiler combines multiple .o files together and adds additional libraries to create the final executable.



Combined Compile-Link

The compiler processes the converts the source file to machine code and then immediately links it.



To –c or not to -c

- How do you control what the compiler does? The much-rumored -c flag.
- If given -c, the compiler will always create a single .o file from a single source file.
- If not given –c, then the compiler will always create an executable using all of the files it's given.
  - This always involves linking, and may involve compilation too if you pass in .cpp files
- Note: there are other types things you can link besides executables, such as static libraries (.a) and shared libraries (.so/.dylib/.dll), but we won't cover those in this class.

Common Compile Options

Option	Function	Notes
`-Wall`	Enables all standard warnings, so the compiler will tell you about code it thinks is suspicious	-W<name> and –Wno-<name> allow you to enable or disable specific warnings
`--std=c++17`	Tells the compiler to use C++17	Can also select older C++ standards such as C++98 and C++11.
`-g`	Generates debugging information so that you can view stacktraces in your code with GDB and Valgrind	May also want to use -O0 to disable optimization for a better debugging experience.
`-I<folder>`	Adds a directory to the #include search path	-include includes a specific header file at the top of each cpp file
`-o <name>`	Specifies the name of the output file.	If not given, by convention the result is named “a.out”

Incremental Builds

- Why don’t we just list all .cpp files in a single compile-link command and forget about all of this?
- Build speed, that’s why.
- C++ files can take a significant amount of time to compile, especially once you have many files that contain lots of code and templates.
- Compiling each cpp file into its own .o file means that instead of rebuilding all sources when you change something, you just rebuild that one .o file, and then redo the final linking.
- This is significantly faster!

Beginning the Lab

- Now that you understand the basics of building C++, we'll now begin the Makefile lab.
- In Part 1, we'll go through constructing a super simple Makefile together.
- Then, in Part 2, it will be up to you to write a makefile for a somewhat larger and more complicated program.