Makefiles

CSCI 3308

Fall 2014

Liz Boese

MAKEFILES

The g++ Compiler

- What happens when you call g++ to build your program?
- Phase 1, **Compilation**:
  .cpp files are compiled into .o object modules

- Phase 2, **Linking**:
  .o modules are linked together to create a single executable.

The Makefile: Input for make

Simplify the compile process:

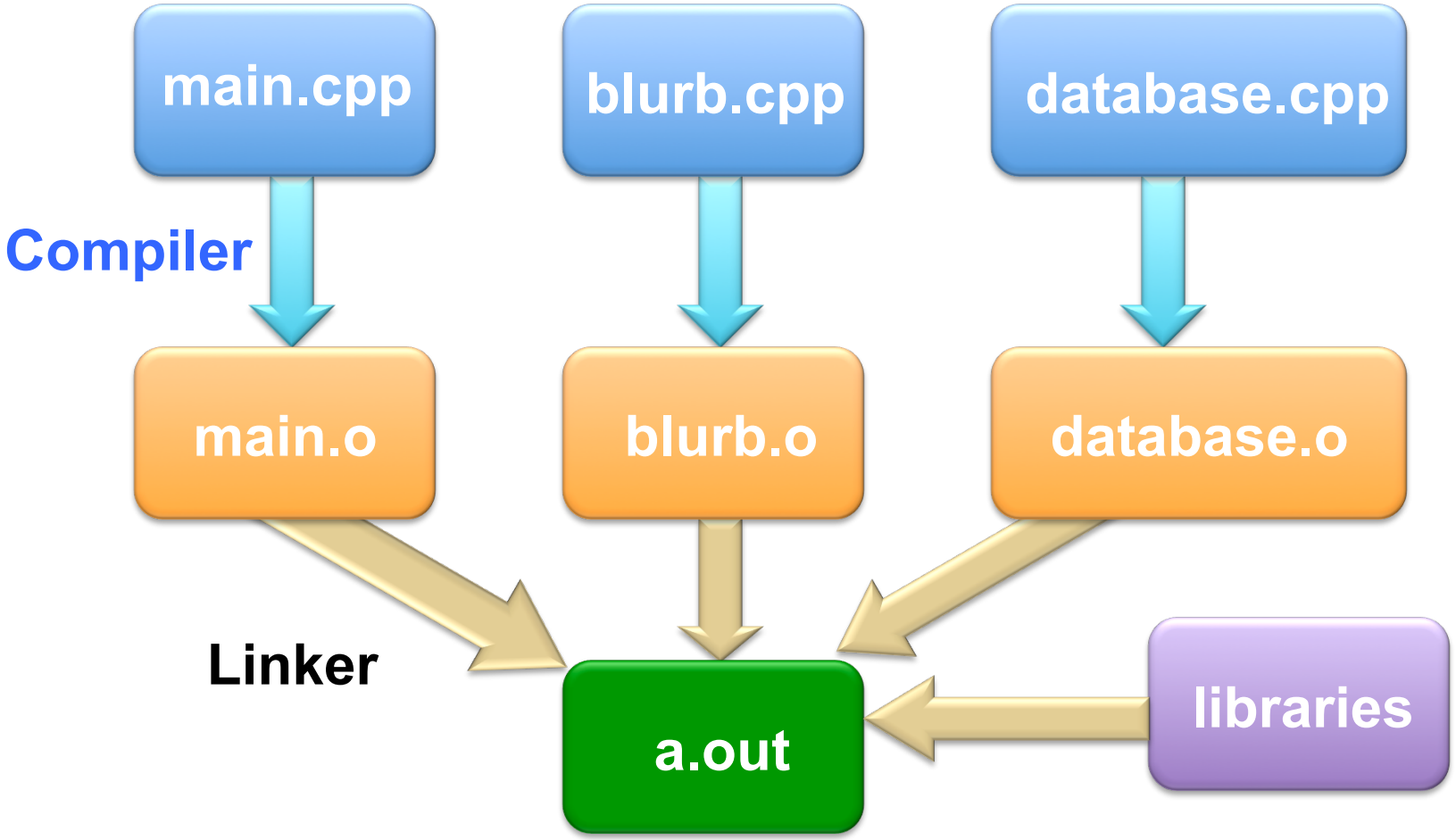
- Create a file called “Makefile”
  - “make” command searches for “Makefile”
- Determines which pieces of a large program need to be compiled or recompiled, and issues commands to compile and link them in an automated fashion.
- Just type “make” at the terminal

Makefiles in Professional C++

- Fast to run: Only recompile what you need to recompile upon code change, with a simple command

- Great for code versioning and deployment; just version the source code and the Makefile and quickly deploy on any compatible system (no need to version binaries).

The Process


main.cpp
blurb.cpp
database.cpp
Compiler
main.o
blurb.o
database.o
Linker
a.out
libraries

How g++ Does All This

- Calling:

```
g++ <option flags> <file list>
```

- When g++ is given .cpp files, it performs both compilation and linking.
- If given .o files, it performs linking only.

A Basic Makefile

- A first Makefile

starts a comment

```
# build an executable named myprog
all: myprog.cpp
	gcc -o myprog myprog.cpp

clean:
	$(RM) myprog
```

The tabs are important!

“clean” Target: Gotta Have One

- An important target that represents an action rather than a g++ operation.
- Has no dependencies, runs a command to remove all the compilation products from the directory, “cleaning” things up.
- Useful for porting code, getting rid of corrupted files, etc.
- Normally also removes the “core” file if it’s present from a past program meltdown.
- Call by typing “make clean” into prompt.

A Basic Makefile

- A first Makefile: Create one for your Student program

```
# build an executable for Student example
all: Student.cpp Driver.cpp
  g++ -std=c++11 -o studs Student.cpp \
     Driver.cpp

clean:
    $(RM) studs
```

A slash \ allows you to go to the next line

Sample Makefile

- Makefiles main element is called a *rule*:

```
target : dependencies
TAB    commands                    #shell commands
```

Example:

```
my_prog : eval.o main.o
    g++ -o my_prog eval.o main.o


eval.o : eval.c eval.h
    g++ -c eval.c
main.o : main.c eval.h
    g++ -c main.c
```

```
# -o to specify
executable file name
# -c to compile only
(no linking)
```

Makefile Lingo

- **Target**: Usually the name of an executable or object file that is generated by g++, but it can also be the name of an action to carry out.
- **Prerequisites**: A list of files needed to create the target. If one of these files have changed, then the make utility knows that it has to build the target file again. Also called **dependencies**.
- **Command**: An action that make carries out, usually a g++ compilation or linking command. In make, commands are run and generate output just as if they were entered one by one into the UNIX prompt.

Options For g++ **Compilation**

- **`-c`**
  Compiles **`.cpp`** file arguments to **`.o`** but does not link *(we'll need this for "make" later).*
- **`-o`**
  Specifies the name of the program to be linked together *(instead of a.out)*
- **`-std=c++11`**
- **`-Wall`**
  Give warnings about code that's probably in error.
  - For example, it will catch and report:

```
int number = GetRandomInteger()
if (number = 6)
   cout << "Always equals 6!";
```

SEPARATING COMPILATION & LINKING

The “textr” App Makefile, Take 1:

```
textr : main.o blurb.o database.o
  g++ -o textr main.o blurb.o database.o
main.o : main.cpp
  g++ -Wall -c main.cpp
blurb.o : blurb.cpp blurb.h
   g++ -Wall -c blurb.cpp
database.o : database.cpp database.h
  g++ -Wall -c database.cpp
clean:
  $(RM) core textr main.o blurb.o database.o
```

Makefile - Student

- Modify your Student Makefile to separate compilation and linking. Name your program

VARIABLES

Next Step: Add Variables

- Add a variable for the object files

```
OBJS = main.o blurb.o database.o

textr : $(OBJS)
        g++ -o textr $(OBJS)
main.o : main.cpp
        g++ -Wall -c main.cpp
blurb.o : blurb.cpp  blurb.h
        g++ -Wall -c blurb.cpp
database.o : database.cpp database.h
        g++ -Wall -c database.cpp
clean:
       rm -f core textr $(OBJS)
```

Makefile - Student

- Modify your Student Makefile to add a variable for all the object files (much better, eh?)

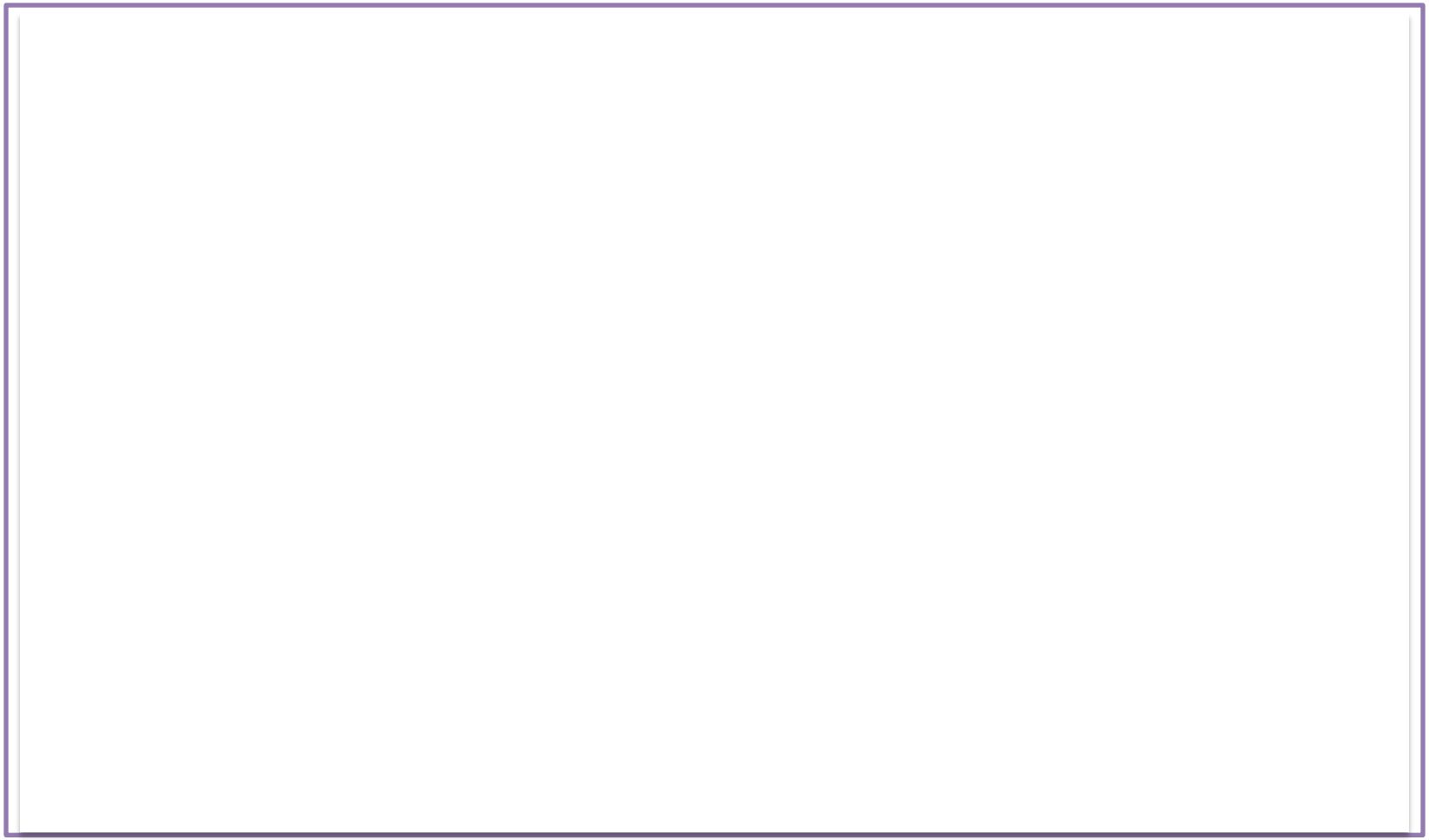
Next Step: Add Variables

- Add a variable for the compiler flags CPPFLAGS

```
OBJS = main.o blurb.o database.o
CPPFLAGS = -Wall
textr : $(OBJS)
        g++ -o textr $(OBJS)
main.o : main.cpp
        g++ $(CPPFLAGS) -c main.cpp
blurb.o : blurb.cpp  blurb.h
        g++ $(CPPFLAGS) -c blurb.cpp
database.o : database.cpp database.h
        g++ $(CPPFLAGS) -c database.cpp
clean:
       rm -f core textr $(OBJS)
```

Makefile - Student

- Modify your Student Makefile to add a variable for the compiler options



Variable for Compiler & Program

```
#Makefile for “textr” C++ application
#Created by Dan Wilson 1/29/06

PROG = textr
CC = g++
CPPFLAGS = -g –Wall
OBJS = main.o blurb.o database.o

$(PROG) : $(OBJS)
        $(CC) -o $(PROG) $(OBJS)
main.o : main.cpp
        $(CC) $(CPPFLAGS) -c main.cpp
blurb.o : blurb.cpp blurb.h
        $(CC) $(CPPFLAGS) -c blurb.cpp
database.o : database.cpp database.h
        $(CC) $(CPPFLAGS) -c database.cpp
clean:
        rm -f core $(PROG) $(OBJS)
```

Makefile - Student

- Add variables for compiler & executable

MAKE IS INTELLIGENT

Next Step: Omit the Obvious!

- (make knows .cpp -> .o)

```
PROG = textr
CC = g++
CPPFLAGS = -g –Wall
OBJS = main.o blurb.o database.o

$(PROG) : $(OBJS)
        $(CC) -o $(PROG) $(OBJS)
main.o :
        $(CC) $(CPPFLAGS) -c main.cpp
blurb.o :  blurb.h
        $(CC) $(CPPFLAGS) -c blurb.cpp
database.o : database.h
        $(CC) $(CPPFLAGS) -c database.cpp
clean:
        rm -f core $(PROG) $(OBJS)
```

Makefile - Student

- Modify your Student Makefile to remove the unnecessary references to .cpp files *(2 places only!)*

TROUBLESHOOTING

Troubleshooting

- If you get this error message, it's because your tab is not correct

```
*** missing separator.  Stop.
```

ANT

Ant

- Java version of a Makefile
- Relies on
  - Targets
  - Dependencies
  - Commands
- Uses XML
- build.xml in project root directory
- From command-line, type:
  ant
- Eclipse and other IDEs have it integrated

Example Ant file

```
<?xml version="1.0"?>
<project name="HelloJavaAnt" default="build">
        <property name="src.dir" location="src" />
        <property name="build.dir" location="build/classes" />
        <target name="mkdir">
                <mkdir dir="${build.dir}" />
                <mkdir dir="${build.dir}/JARs" />
        </target>
        <target name="build" depends="mkdir">
                <javac srcdir="${src.dir}"
                   destdir="${build.dir}"/>
        </target>
        <target name="compress" depends="build">
                <jar destfile="${build.dir}/JARs"
                   basedir="${build.dir}"
        </target>
</project>
```