C++ Programming on Linux Multi-file development

CS 2308 Spring 2017

Jill Seaman

Programs with Multiple Files

- How the code is usually split up
 - Put main in its own file, with helper functions
 acts like a driver
 - Put each class declaration in a separate *.h file (called a header file)
 - * Put the implementation of each class (the member function definitions) in its own *.cpp file
 - * Each *.cpp file (even the driver) must #include (directly or indirectly) the header file (*.h) of each class that it uses or implements.

2

* NEVER #include *.cpp files!!!

Time class, separate files

Time.h	Driver.cpp
<pre>#include <string> using namespace std;</string></pre>	<pre>//Example using Time class #include<iostream> #include "Time.h"</iostream></pre>
<pre>// models a 12 hour clock class Time {</pre>	using namespace std;
	<pre>int main() {</pre>
private:	Time t;
int hour;	t.setHour(12);
int minute;	<pre>t.setMinute(58);</pre>
<pre>void addHour();</pre>	<pre>cout << t.display() <<endl; t.addMinute();</endl; </pre>
public:	<pre>cout << t.display() << endl;</pre>
void setHour(int);	t.addMinute();
<pre>void setMinute(int);</pre>	<pre>cout << t.display() << endl;</pre>
int getHour() const;	return 0;
<pre>int getMinute() const;</pre>	}
<pre>string display() const; void addMinute();</pre>	
};	

Time class, separate files

Time.cpp	
<pre>#include <iomanip></iomanip></pre>	<pre>void Time::addHour() {</pre>
<pre>#include <sstream></sstream></pre>	if (hour == 12)
<pre>#include "Time.h"</pre>	hour = 1;
using namespace std;	else
	hour++;
<pre>void Time::setHour(int hr) {</pre>	}
hour = hr;	<pre>void Time::addMinute() {</pre>
}	if (minute == 59) {
	minute = 0;
<pre>void Time::setMinute(int min) {</pre>	addHour();
minute = min;	} else
}	<pre>minute++;</pre>
	}
<pre>int Time::getHour() const {</pre>	<pre>string Time::display() const {</pre>
return hour;	<pre>string hourStr = to_string(hour);</pre>
}	<pre>string minuteStr = to_string(minute);</pre>
int mine (antificate () and (if (minuteStr.length()==1)
<pre>int Time::getMinute() const { return minute;</pre>	<pre>minuteStr = "0" + minuteStr; meturn hourStr "" minuteStr;</pre>
return minute;	return hourStr + ":" + minuteStr;
3	3
	4

How to compile a multiple file program

- From the command line (files in either order):
 - [...]\$g++ Time.cpp Driver.cpp
 - The header file should **not** be listed. (it is #included in *.cpp files)
 - * one (and only one) file must have the main function
- a.out is (by default) the executable file for the entire program.

[...]\$./a.out 12:58 12:59 1:00

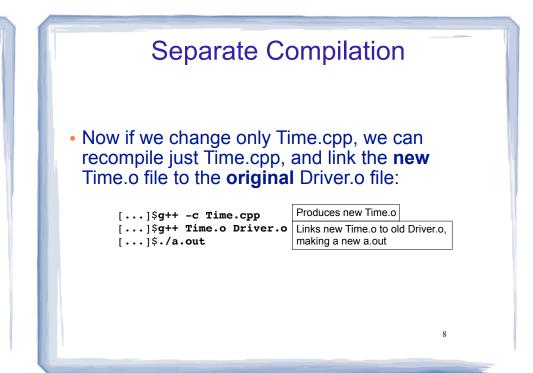
5

Separate Compilation

- If we make a change to Driver.cpp, we have to recompile it
 - * but perhaps we would rather not have to recompile Time.cpp as well.
- We can compile one file at a time, and **link** the results together later to make the executable.
- Compiling without linking (use -c option):

[...]\$g++ -c Time.cpp [...]\$g++ -c Driver.cpp

 c option produces <u>object files</u>, with a .o extension (Time.o, Driver.o)



Make

- <u>Make</u> is a utility that manages (separate) compilation of large groups of source files.
- After the first time a project is compiled, <u>make</u> re-compiles **only the changed files** (and the files depending on the changed files).
- These dependencies are defined by rules contained in a makefile.
- The rules are defined and managed by humans (programmers).

Make

• Rule format:

target: [prerequisite files]
<tab>[linux command to execute]

- target is a filename (or an action/goal name)
- In order to produce the target file, the prerequisite files must exist and be up to date (if not, make finds a rule to produce them).
- An example rule:

Time.o: Time.cpp Time.h g++ -c Time.cpp If Time.o does not exist, OR if Time.cpp or Time.h is **newer** than Time.o, (re)produce Time.o using this command

Makefile

• The makefile is a text file named "makefile":

#makefile

a.out: Driver.o Time.o g++ Driver.o Time.o

Driver.o: Driver.cpp Time.h g++ -c Driver.cpp

Time.o: Time.cpp Time.h g++ -c Time.cpp You can use nano to create this file

9

Do **not** copy/paste this to your makefile,

Don't forget the tabs

11

Don't call it makefile.txt

Make

- running make from the linux/unix prompt with no arguments executes first rule in the makefile.
 - * This may trigger execution of other rules.

[...]\$ make

• executing the make command followed by a target executes the rule for that target.

[...]\$ make Time.o

