

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 (including the driver) must #include the header file of each class that it uses or implements.

2

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>
// models a 12 hour clock	using namespace std;
class Time {	<pre>int main() {</pre>
private:	Time t;
int hour;	t.setHour(12);
int minute;	t.setMinute(58);
<pre>void addHour();</pre>	<pre>cout << t.display() <<endl; t.addMinute();</endl; </pre>
public:	<pre>cout << t.display() << endl;</pre>
<pre>void setHour(int);</pre>	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>	
#include <sstream></sstream>	if (hour == 12)	
#include "Time.h"	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>	
int mine antitions() see at (}	
<pre>int Time::getHour() const {</pre>	<pre>string Time::display() const {</pre>	
return hour;	ostringstream sout;	
}	<pre>sout.fill('0'); sout << hour << ":"</pre>	
<pre>int Time::getMinute() const {</pre>	<pre>sour << nour << :</pre>	
return minute;		
	return sout.str();	
}	}	
	4	

How to compile a multiple file program

• From the command line (either order):

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

- The header file does not need to be listed. (it only needs to be #included in *.cpp files)
- * one file must have the main function
- a.out is (by default) the executable for the entire program.

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

Separate Compilation

- If we make a change to Driver.cpp, we have to recompile it
- * but 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 get 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)

Separate Compilation

• The .o files must be linked together to produce the executable file (a.out):

[...]\$ g++ Time.o Driver.o [...]\$./a.out

r.o Note there is no option used here

5

 Now if we change only Time.cpp, we can recompile just Time.cpp, and link the new .o file to the original Driver.o file:

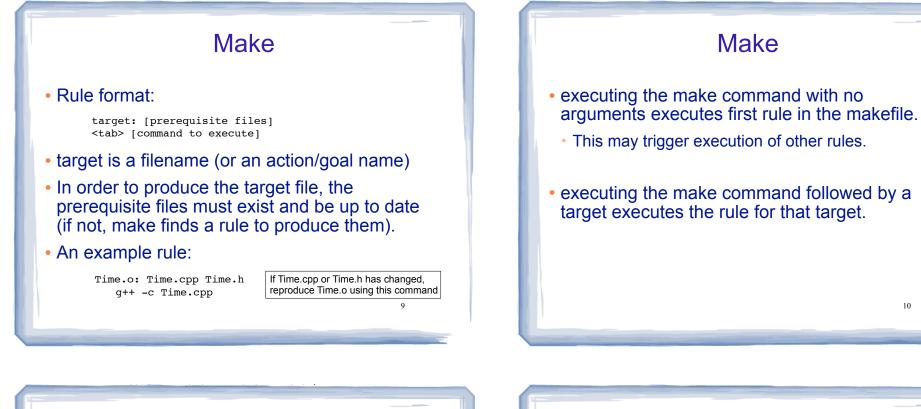
 [...]\$g++ -c Time.cpp
 Produces new Time.o

 [...]\$g++ Time.o Driver.o
 Links new Time.o to old Driver.o, making a new a.out

Make

- Make is a utility that manages (separate) compilation of large groups of source files.
- After the first time a project is compiled, make 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).

8





Compile class + driver using make

- Make: [...]\$ make g++ -c Driver.cpp q++ -c Time.cpp q++ Driver.o Time.o -o timeTest
- Execute: [...]\$./timeTest 12:58 12:59 1:00
- Modify Driver.cpp, make again:

[...]\$ make g++ -c Driver.cpp q++ Driver.o Time.o -o timeTest 10