Multi-file Development
using C++, Linux and Make

CS 3358
Spring 2015

Jill Seaman

Assumptions: What you should already know how to do with Linux

- How to use linux from the command line (basic commands).
- Basic file editing on a linux machine.
- Compile and execute a single file:

```
[...]$g++ hello.cpp
[...]$./a.out
```

- Remote access (secure shell, file transfer)
- CS department lab webpage has documentation on these tasks (Lab tutorials, handouts).
  - or see my CS2308 Linux lecture

C++ 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
  - Put the implementation of each class (the definitions of the member functions) in its own *.cpp file
  - Each file must #include (directly or indirectly) the header file of each class that it uses.

time.h (header file)

```
// file time.h
#include <string>
using namespace std;

  class Time         //new data type
  {
    // models a 12 hour clock
    private:
      int hour;
      int minute;
      void addHour();

    public:
      void setHour(int);
      void setMinute(int);

      string display();
      void addMinute();

  };
```

time.cpp (implementation file)

```
// file time.cpp
#include <sstream>
#include <iomanip>
using namespace std;

#include "time.h"

void Time::setHour(int hr) {
  hour = hr;            // hour is a member var
}
void Time::setMinute(int min) {
  minute = min;         // minute is a member var
}

void Time::addHour() {  // a private member func
  if (hour == 12)
     hour = 1;
  else
     hour++;
}                            //continued . . .
```


time.cpp (implementation file, cont.)

```
void Time::addMinute()
{
  if (minute == 59) {
     minute = 0;
     addHour();   // call to private member func
  } else
     minute++;
}

string Time::display()
// returns time in string formatted to hh:mm
{
   ostringstream sout;
   sout.fill('0');
   sout << hour << ":" << setw(2) << minute;
   return sout.str();
}
```


driver.cpp: A program that uses Time

```
//using Time class (driver.cpp)
#include<iostream>
#include "time.h"
using namespace std;

int main() {
  Time t;
  t.setHour(12);
  t.setMinute(58);
  cout << t.display() <<endl;
  t.addMinute();
  cout << t.display() << endl;
  t.addMinute();
  cout << t.display() << endl;
  return 0;
}
```


How to compile and run 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.
  - 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

- Compiling to intermediate files:

```
[...]$g++ -c time.cpp
[...]$g++ -c driver.cpp
```

  - -c option produces object files, with a .o extension (time.o, driver.o)
- To link the object files into the executable (a.out):

```
[...]$ g++ time.o driver.o
```

- Now if we change only time.cpp, to recompile:

```
[...]$g++ -c time.cpp
[...]$g++ time.o driver.o
```

It re-uses the driver.o produced in the first step

Make

- Make is a utility that manages (separate) compilation of large groups of source files.
- Goal: After the first time a project is compiled, it only re-compiles the newly changed files (and the files depending on the changed files).
- The 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> [command to execute to produce target]
```

- target is a filename (or an action/goal name)
- An example rule:

```
time.o: time.cpp time.h
   g++ -c time.cpp
```

- make command with no arguments executes first rule in makefile.
- make command followed by a target executes the rule for that target.

Makefile

- makefile:

```
#makefile

timeTest: driver.o time.o
   g++ driver.o time.o -o timeTest

driver.o: driver.cpp time.h
   g++ -c driver.cpp

time.o: time.cpp time.h
   g++ -c time.cpp
```

- Note: “timeTest” is the name of the executable file in this example (not a.out).

Compile class + driver using make

- Make:

```
[...]$ make
g++ -c driver.cpp
g++ -c time.cpp
g++ driver.o time.o -o timeTest
```

- Execute:

```
[...]$ ./timeTest
12:58
12:59
1:00
```

- Modify driver.cpp and make again:

```
[...]$ make
g++ -c driver.cpp
g++ driver.o time.o -o timeTest
```

Note that time.cpp is NOT compiled this time.