C++ Programming on Linux

Part 2

CS 2308
Fall 2011

Jill Seaman

Lecture 7

Programs with Multiple Files

- Why split code into multiple files?
  - Separate compilation of files: can speed up compilation when only 1 file is changed.
  - Better organization, easier to find things
  - Facilitates code reuse and sharing
  - Allows multiple programmers to work on the same program.

How to Split C++ Code

- Often put main in its own file
  - setup and call other functions, like a driver
- Put functions that interact with each other in their own file (a sub-system).
- Put functions that are used by many other files/ functions in their own file (utilities: sort, search)
- Note: if a function is called from another file, its prototype must occur in that file, before the function is called.

Header Files

- Problem: prototype for a given function occurs in multiple files, if it is used often.
  - Difficult to maintain if prototype changes
- Convention:
  - put the prototypes of the functions from a given file in another file, called a header file.
  - ex: sprite.cpp and sprite.h
  - #include the header file in every other file that calls one of these functions.
  - header files also contain other common definitions: structures, constants, etc.

Header Files as Interface

- Header file also acts as an *interface* between the users of the functions and their implementation.
- This hides the details from the users.
- Also isolates the users of the functions from changes in implementation of the functions.
- Good Practice: comment the function prototypes in the header file with information about how to use the function, independently of how it is implemented.

Simple Example

- filehello.h

```
// header file for filehello

#include<fstream>
#include<iomanip>

void filehello();
```

- filehello.cpp

```
//hello world from file
//coded by Carol Hazlewood
//September 9, 2009

#include "filehello.h"

using namespace std;

void filehello()
{
    ofstream outFile;
    outFile.open("hello.txt");
    outFile <<  "hello, world." ;
    outFile.close();
}
```

Simple Example

- multhello.cpp

```
// Hello World example 2.  Shows multiple file organization,
// use of header files, and sample makefile.

//hello world
//coded by Carol Hazlewood
//September 9, 2008

#include<iostream>
#include "filehello.h"

using namespace std;

int main()
{

    filehello();

    cout << "Hello, World!" << endl;

    return 0;
}
```


How to compile a multiple file program

- From the command line (either order):

```
[...]$g++ filehello.cpp multhello.cpp
```

  - The header file does not need to be listed. It only needs to be #included.

- a.out is the executable for the entire program.

```
[...]$ ./a.out
Hello, World!
```

Separate Compilation

- Compiling to intermediate files:

```
[...]$g++ -c filehello.cpp
[...]$g++ -c multhello.cpp
```

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

```
[...]$ g++ multhello.o filehello.o
```

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

```
[...]$g++ -c filehello.cpp
[...]$g++ multhello.o filehello.o
```


Make

- Make is a utility that manages compilation of large groups of source files.
- After the first time a project is compiled, it only re-compiles the 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]
```

- target is a filename, or an action/goal name
- make with no arguments executes first rule in makefile.
- make followed by a target executes the rule for that target.
- An example rule:

```
multhello.o: multhello.cpp filehello.h
   g++ -c multhello.cpp
```


Simple Example

- makefile

```
#makefile for hello world with multiple files

multhello:  multhello.o filehello.o
   g++ multhello.o filehello.o -o multhello

multhello.o: multhello.cpp filehello.h
   g++ -c multhello.cpp

filehello.o:  filehello.cpp filehello.h
   g++ -c filehello.cpp

clean:
   rm *.o
   rm multhello
```