

EECS 22: Advanced C Programming

Lecture 11

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 11: Overview

- Warm-up Quiz
- Shared Libraries
 - Static object file archives
 - Dynamically loaded shared objects
- Make and Makefile
 - Rules
 - Dependencies
 - Application example `PhotoLab2`
 - Modules `FileIO`, `Age`, `Main`
 - `Makefile`
 - Advanced features

Quiz: Question 1


- Which Linux command shows you the path to the current directory?
 - a) `cd`
 - b) `pwd`
 - c) `dir`
 - d) `ls`
 - e) `list`

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

3

Quiz: Question 1

- Which Linux command shows you the path to the current directory?
 - a) `cd`
 -  b) `pwd`
 - c) `dir`
 - d) `ls`
 - e) `list`

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

4

Quiz: Question 2


- Which of the following Linux commands renames file “homework1.c” into “text1.c”?
 - a) `rn text1.c homework1.c`
 - b) `rn homework1.c text1.c`
 - c) `rm text1.c homework1.c`
 - d) `mv homework1.c text1.c`
 - e) `mv text1.c homework1.c`

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

5

Quiz: Question 2

- Which of the following Linux commands renames file “homework1.c” into “text1.c”?
 - a) `rn text1.c homework1.c`
 - b) `rn homework1.c text1.c`
 - c) `rm text1.c homework1.c`
 -  d) `mv homework1.c text1.c`
 - e) `mv text1.c homework1.c`

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

6

Quiz: Question 3


- What is C *not*?
 - a) a structured programming language
 - b) a object-oriented programming language
 - c) a compiled programming language
 - d) a high-level programming language
 - e) a portable programming language

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

7

Quiz: Question 3

- What is C *not*?
 - a) a structured programming language
 -  b) a object-oriented programming language
 - c) a compiled programming language
 - d) a high-level programming language
 - e) a portable programming language

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

8

Quiz: Question 4

- What is the meaning of the following code fragment?


```
/* printf("C programming is great!\n") */
```

- a) it prints "C programming is boring!"
- b) it prints "C programming is great!"
- c) it is a syntax error because a semicolon is missing after the `printf()` statement
- d) it is the main function of the C program
- e) it is a comment ignored by the compiler

Quiz: Question 4

- What is the meaning of the following code fragment?

```
/* printf("C programming is great!\n") */
```

- a) it prints "C programming is boring!"
- b) it prints "C programming is great!"
- c) it is a syntax error because a semicolon is missing after the `printf()` statement
- d) it is the main function of the C program
-  e) it is a comment ignored by the compiler

Quiz: Question 5

- What is true about of the following compiler call? (Check all that apply!)

```
% gcc HelloWorld.c -Wall -ansi -o HelloWorld
```

- a) the GNU C Compiler is called to generate an executable program called `HelloWorld`
- b) the compiler will print warning and/or error messages about any non-ANSI compliance in the code
- c) the compiler will ignore all warnings
- d) the compiler will read the file `HelloWorld.c`
- e) the compiler will overwrite the `HelloWorld` file if it already exists

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

11

Quiz: Question 5

- What is true about of the following compiler call? (Check all that apply!)

```
% gcc HelloWorld.c -Wall -ansi -o HelloWorld
```

- a) the GNU C Compiler is called to generate an executable program called `HelloWorld`
- b) the compiler will print warning and/or error messages about any non-ANSI compliance in the code
- c) the compiler will ignore all warnings
- d) the compiler will read the file `HelloWorld.c`
- e) the compiler will overwrite the `HelloWorld` file if it already exists

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

12

Shared Libraries

- When a set of modules is useful for multiple applications, a *shared library* can be built for reusing the functionality provided by the modules
 - 1) *Static library*: Archive of object files (linked at compile-time)
 - Examples: `libc.a`, `libM.a`
 - Archive tools `ar` and `ranlib`
 - can combine multiple object files into linker archives
 - 2) *Dynamic library*: Shared object file (loaded at run-time)
 - Examples: `libc.so`, `libM.so`
 - Compiler `gcc`
 - can create position-independent code (`-fPIC`)
 - can create shared libraries (`-shared`)
 - can create executable files that load shared object files at run-time (`-Lpath -Xlinker -R -Xlinker path`)

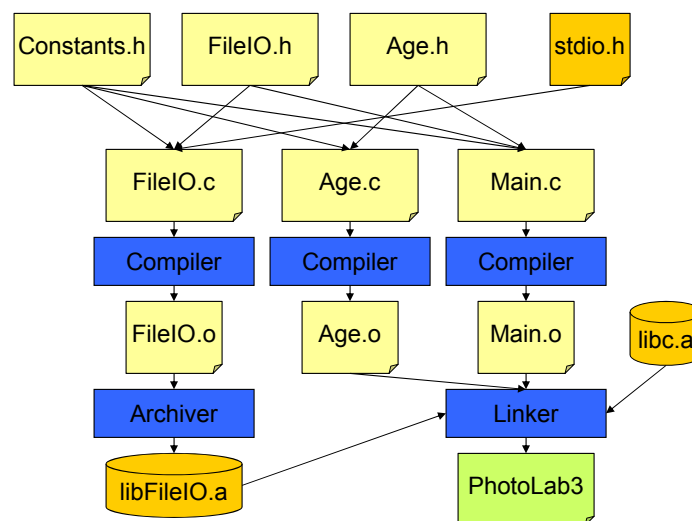
EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

13

Shared Libraries

- Example: Shared `FileIO` Library for `PhotoLab3`



EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

14

Shared Libraries

- Example: Shared `FileIO` Library for `PhotoLab3`
- Option 1: Static shared library
 - Preparing the archive file:

```
% vi Constants.h % gcc -c FileIO.c -o FileIO.o -Wall -ansi
% vi FileIO.h % ar rc libFileIO.a FileIO.o
% vi FileIO.c % ranlib libFileIO.a
%
```

- Building and compiling the application:

```
% vi Age.h % gcc -c Age.c -o Age.o -Wall -ansi
% vi Age.c % gcc -c Main.c -o Main.o -Wall -ansi
% vi Main.c % gcc Main.o Age.o -lFileIO -L. -o PhotoLab3
% PhotoLab3
%
```

Shared Libraries

- Example: Shared `FileIO` Library for `PhotoLab3`
- Option 2: Dynamically loaded shared library
 - Preparing the shared object file:

```
% gcc -c -fPIC FileIO.c -o FileIO.o -Wall -ansi
% gcc -shared -fPIC -o libFileIO.so FileIO.o
%
```

- Building and compiling the application:

```
% gcc -c Age.c -o Age.o -Wall -ansi
% gcc -c Main.c -o Main.o -Wall -ansi
% gcc Main.o Age.o -lFileIO -L.
  -Xlinker -R -Xlinker . -o PhotoLab3
% PhotoLab3
% ldd PhotoLab3
    linux-vdso.so.1 => (0x00007fff07550000)
    libFileIO.so => ./libFileIO.so (0x00007f2ddf1ac000)
    libc.so.6 => /lib64/libc.so.6 (0x00000035f5000000)
    /lib64/ld-linux-x86-64.so.2 (0x00000035f4c00000)
%
```


Make and Makefile

- Building an application from multiple source files requires multiple compiler calls
 - Typing compiler calls is tedious
 - Automation can help: build scripts!
- Linux tool **make**
 - reads compilation rules from a **Makefile** (script)
 - automatically executes necessary build commands
- **Makefile** consists of a set of rules
- Rules consist of
 - *Target* (usually a file)
 - *Dependencies* (typically source files)
 - *Command(s)* to generate the target from the sources

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

17

Make and Makefile

- Example Rule:

```
HelloWorld: HelloWorld.c
    gcc -Wall -ansi HelloWorld.c -o HelloWorld
```

Target **HelloWorld**

- depends on source file **HelloWorld.c**
- can be built by executing the listed command
 - Note: Command line starts with a *horizontal tabulator* (TAB)!

- Compilation: Make!

```
% vi HelloWorld.c
% vi Makefile
% make
gcc -Wall -ansi HelloWorld.c -o HelloWorld
% HelloWorld
Hello World!
%
```

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

18

Make and Makefile

- **Makefile** with Multiple Rules
 - **make**
 - Builds the first target (executes first rule)
 - **make target**
 - Builds the specified target (executes specified rule)
- **Rules and Dependencies**
 - Rules are applied *if and only if* the target file...
 - ... does not exist, or
 - ... is out of date
 - Target file is older than any of its dependencies
 - Every file has a time stamp of its last modification time, so **make** can determine what needs to be re-built
 - Rules are applied *recursively*
 - If a dependency is missing or is not up-to-date, **make** builds it first!

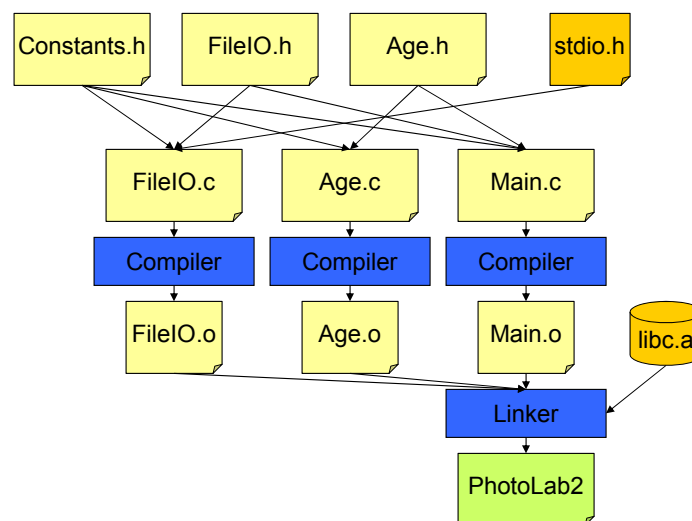
EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

19

Make and Makefile

- **Making PhotoLab2**



EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

20

Make and Makefile

- Example: Basic Makefile for PhotoLab2

```
# Makefile: PhotoLab2

PhotoLab2: FileIO.o Age.o Main.o
    gcc -Wall -ansi FileIO.o Age.o Main.o -o PhotoLab2

FileIO.o: FileIO.c FileIO.h Constants.h
    gcc -Wall -ansi -c FileIO.c -o FileIO.o

Age.o: Age.c Age.h Constants.h
    gcc -Wall -ansi -c Age.c -o Age.o

Main.o: Main.c Constants.h FileIO.h Age.h
    gcc -Wall -ansi -c Main.c -o Main.o
```

Make and Makefile

- Example session: `make PhotoLab2`

```
% vi Constants.h
% vi FileIO.h
% vi FileIO.c
% vi Age.h
% vi Age.c
% vi Main.c
% vi Makefile
% make
gcc -Wall -ansi -c FileIO.c -o FileIO.o
gcc -Wall -ansi -c Age.c -o Age.o
gcc -Wall -ansi -c Main.c -o Main.o
gcc -Wall -ansi FileIO.o Age.o Main.o -o PhotoLab2
% PhotoLab2
% vi Age.c
% make
gcc -Wall -ansi -c Age.c -o Age.o
gcc -Wall -ansi FileIO.o Age.o Main.o -o PhotoLab2
% PhotoLab2
```

Advanced Makefile

- Commands issued by **make**
 - Command line must start with a (horizontal) TAB character
 - Spaces are not recognized!
 - Multiple commands are executed in order
 - Long command lines can be wrapped to the next line by a backslash (\) immediately followed by a newline
 - Commands are echoed to the standard output
 - Echo can be suppressed with a @ prefix before the command
 - Commands are executed as shell commands
 - The **sh** shell is used (similar to **bash** in Linux)
 - Return value of command determines success
 - Return value 0 indicates success (no errors)
 - Return value not equal to 0 indicates error
 - Execution of commands stops with the first error
 - Errors can be ignored with a - prefix before the command

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

23

Advanced Makefile

- Advanced Features
 - Makefile variables


```
DEBUG = -g -DDEBUG
CFLAGS = -Wall -ansi $(DEBUG)
...
Program: Program.c Program.h
gcc $(CFLAGS) Program.c -o Program
```
 - Dummy targets (aka. *pseudo* or *phony* targets)


```
# default target
all: PhotoLab2

clean:
    rm -f *.o
    rm -f PhotoLab2
```
 - Many more features are available...
 - **man make**

EECS22: Advanced C Programming, Lecture 11

(c) 2016 R. Doemer

24

Advanced Makefile

- Advanced Makefile for PhotoLab2 (part 1/2)

```
# Makefile: PhotoLab2
# 10/23/12 RD

# variable definitions
CC      = gcc
DEBUG   = -g -DDEBUG
#DEBUG  = -O2 -DNDEBUG
CFLAGS  = -Wall -ansi $(DEBUG) -c
LFLAGS  = -Wall $(DEBUG)

# convenience targets
all: PhotoLab2

clean:
    rm -f *.o
    rm -f PhotoLab2

...
```

Advanced Makefile

- Advanced Makefile for PhotoLab2 (part 2/2)

```
...

# compilation rules
FileIO.o: FileIO.c FileIO.h Constants.h
    $(CC) $(CFLAGS) FileIO.c -o FileIO.o

Age.o: Age.c Age.h Constants.h
    $(CC) $(CFLAGS) Age.c -o Age.o

Main.o: Main.c Constants.h FileIO.h Age.h
    $(CC) $(CFLAGS) Main.c -o Main.o

PhotoLab2: FileIO.o Age.o Main.o
    $(CC) $(LFLAGS) FileIO.o Age.o Main.o -o PhotoLab2
```