EECS 22: Advanced C Programming
Lecture 8

Rainer Dömer

doemer@uci.edu

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

Lecture 8: Overview

• Warm-up Quiz

• Make and Makefile
  – Rules
  – Dependencies
  – Application example **PhotoLab2**
    • Module **FileIO**
    • Module **Age**
    • Module **Main**
    • **Makefile**
  – Advanced features

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   2

Quiz: Question 1

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

Quiz: Question 1

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

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`

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`

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 8   (c) 2014 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 8   (c) 2014 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

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   9

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

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   10

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 8   (c) 2014 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 8   (c) 2014 R. Doemer   12

Quiz: Question 6

- Which of the following constructs is a valid binary operator in C?
  (Check all that apply!)
  a) /
  b) %
  c) !
  d) @
  e) >>

Quiz: Question 6

- Which of the following constructs is a valid binary operator in C?
  (Check all that apply!)
  a) /
  b) %
  c) !
  d) @
  e) >>

Quiz: Question 7

- What is the value of the integer **x** after the following statement?

```
x = 11 / 3 + 11 % 3;
```

a) 1
b) 2
c) 3
d) 4
e) 5

Quiz: Question 7

- What is the value of the integer **x** after the following statement?

```
x = 11 / 3 + 11 % 3;
```

a) 1
b) 2
c) 3
d) 4
e) 5

Quiz: Question 8

- What is the value of the variable **x** after the following lines of code?

```
unsigned char x = 42;

x += 1024;
if (x < 0)
    { x = 10; }
if (x > 255)
    { x = 20; }
```

a) 0
b) 10
c) 20
d) 42
e) 1066

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   17

Quiz: Question 8

- What is the value of the variable **x** after the following lines of code?

```
unsigned char x = 42;

x += 1024;
if (x < 0)
    { x = 10; }
if (x > 255)
    { x = 20; }
```

a) 0
b) 10
c) 20
→ d) 42
e) 1066

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   18

Quiz: Question 9

- Which of the following program fragments will *not* terminate? (Check all that apply!)

a)
```
int a = 1;
while(a < 1000000)
    { a++; }
```

```
int a = 10;
while(a > 0)
    { a = a / 3; }
```

b)
```
int a = 0;
while(a < 1000)
    { a = a * 3; }
```

d)
```
int a = 1;
while(a < 1000)
    { a = a << 1; }
```

c)
```
int a = 1;
while(a == 1)
    { a = a % 10; }
```

e)

Quiz: Question 9

- Which of the following program fragments will *not* terminate? (Check all that apply!)

a)
```
int a = 1;
while(a < 1000000)
    { a++; }
```

```
int a = 10;
while(a > 0)
    { a = a / 3; }
```

b) →
```
int a = 0;
while(a < 1000)
    { a = a * 3; }
```

d)
```
int a = 1;
while(a < 1000)
    { a = a << 1; }
```

c) →
```
int a = 1;
while(a == 1)
    { a = a % 10; }
```

e)

Quiz: Question 10

- Given two global variables `int x=7` and `int y=8`, which of the following functions properly swaps the values such that `x=8` and `y=7`? (Check all that apply!)

a)
```
void swap(int x, int y)
{ x = y; y = x;
}
```

b)
```
void swap(void)
{ x = y; y = x;
}
```

c)
```
void swap(void)
{ int t;
   t = x; x = y; y = t;
}
```

d)
```
void swap(void)
{ int t;
   t = y; y = x; x = t;
}
```

e)
```
void swap(int x, int y)
{ int t;
   t = x; x = y; y = t;
}
```

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   21

Quiz: Question 10

- Given two global variables `int x=7` and `int y=8`, which of the following functions properly swaps the values such that `x=8` and `y=7`? (Check all that apply!)

a)
```
void swap(int x, int y)
{ x = y; y = x;
}
```

b)
```
void swap(void)
{ x = y; y = x;
}
```

c) ⬅
```
void swap(void)
{ int t;
   t = x; x = y; y = t;
}
```

d) ⬅
```
void swap(void)
{ int t;
   t = y; y = x; x = t;
}
```
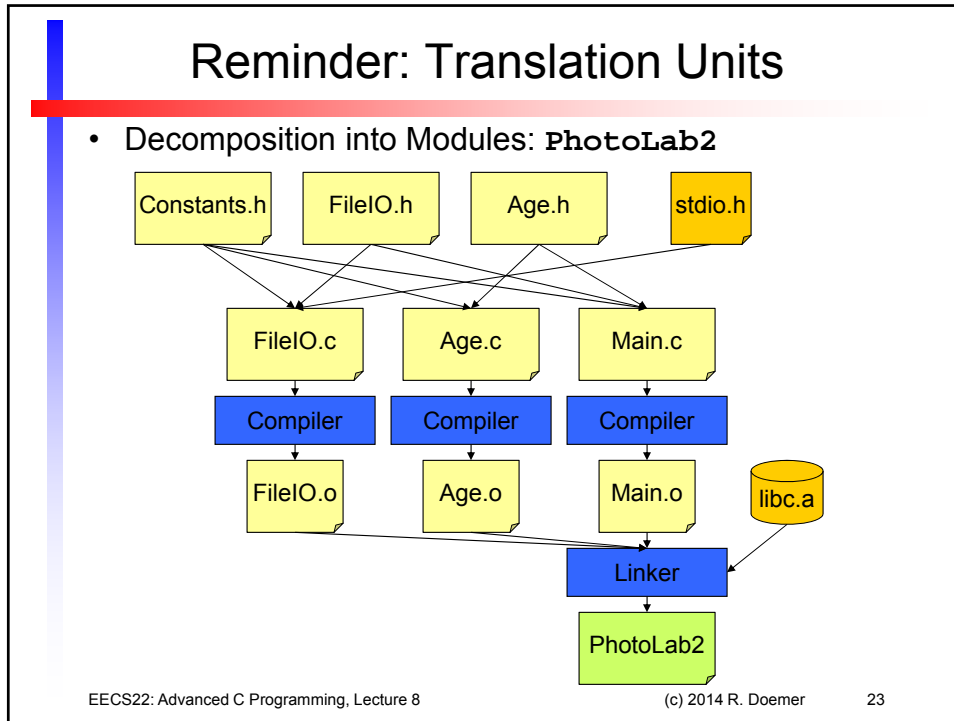
e)
```
void swap(int x, int y)
{ int t;
   t = x; x = y; y = t;
}
```

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   22

Reminder: Translation Units

- Decomposition into Modules: **PhotoLab2**

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   23

Reminder: Application Example

- Build and compilation session:

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

RingMall.ppm   RingMall1965.ppm

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   **24**

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 8   (c) 2014 R. Doemer   25

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 8   (c) 2014 R. Doemer   26

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!
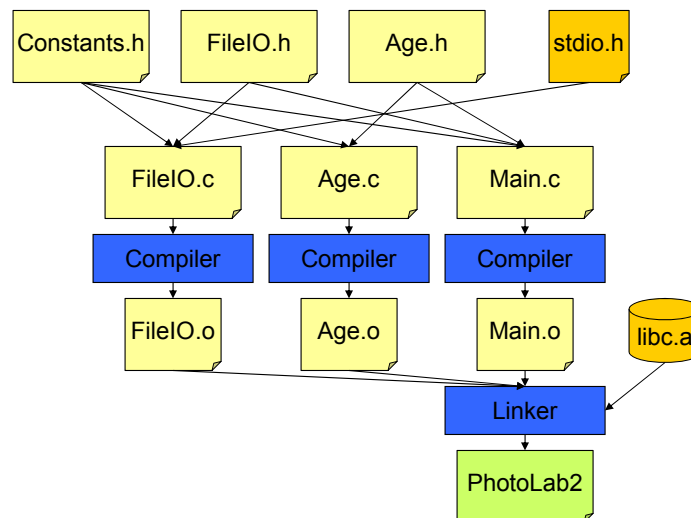
EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   27

Application Example

- Making **PhotoLab2**

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   28

Application Example

- 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
```

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   29

Application Example

- 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
```

EECS22: Advanced C Programming, Lecture 8   (c) 2014 R. Doemer   **30**

Make and 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 8   (c) 2014 R. Doemer   31

Make and 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 8   (c) 2014 R. Doemer   32

Application Example

- 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

...
```

Application Example

- 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
```