

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, Makefile
 - Rules
 - Dependencies
 - **Makefile** for application example **PhotoLab2**
 - Module **FileIO**
 - Module **Age**
 - Module **Main**
 - 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 8

(c) 2011 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 8

(c) 2011 R. Doemer

4

Quiz: Question 2


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

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

5

Quiz: Question 2

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

EECS22: Advanced C Programming, Lecture 8

(c) 2011 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 8

(c) 2011 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) 2011 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 8

(c) 2011 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) 2011 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) >>

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

13

Quiz: Question 6

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

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

14

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 integer `x` after the following statement?


```
x = (10 - (3 - (20 - -10)));
```

- a) 7
- b) 17
- c) 27
- d) 37
- e) 77

Quiz: Question 8

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

```
x = (10 - (3 - (20 - -10)));
```

- a) 7
- b) 17
- c) 27
-  d) 37
- e) 77

Quiz: Question 9


- Which of the following format strings will print an **unsigned long** value in decimal format when used with `printf()`?
 - a) `"%u"`
 - b) `"%ud"`
 - c) `"%d"`
 - d) `"%lu"`
 - e) `"%ui"`

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

19

Quiz: Question 9

- Which of the following format strings will print an **unsigned long** value in decimal format when used with `printf()`?
 - a) `"%u"`
 - b) `"%ud"`
 - c) `"%d"`
 -  d) `"%lu"`
 - e) `"%ui"`

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

20

Quiz: Question 10


- Which of the following statements will correctly read a decimal value from `stdin` into a variable `x` of type `signed int`?
 - a) `stdin("%x", &u);`
 - b) `stdin("%u", x);`
 - c) `scanf("%d", &x);`
 - d) `scanf("&x", %u);`
 - e) `scanf("&x", %d);`

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

21

Quiz: Question 10

- Which of the following statements will correctly read a decimal value from `stdin` into a variable `x` of type `signed int`?
 - a) `stdin("%x", &u);`
 - b) `stdin("%u", x);`
 -  c) `scanf("%d", &x);`
 - d) `scanf("&x", %u);`
 - e) `scanf("&x", %d);`

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

22

Make, Makefile

- Building an application from multiple source files requires multiple compiler calls
 - Typing compiler calls is tedious
 - Automation can help: build scripts!
- *Make* tool
 - reads compilation rules from a *Makefile* (script)
 - executes necessary build commands automatically
- *Makefile* consists 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) 2011 R. Doemer

23

Make, 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 command listed
 - 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) 2011 R. Doemer

24

Make, 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 target file
 - Target does not exist, or
 - Target file is out of date
 - Target file is older than any of its dependencies
 - Every file has a time stamp of its last modification time
 - Rules are applied *recursively*
 - If a dependency is not up-to-date, make builds it first!

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
```

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) 2011 R. Doemer

27

Make, Makefile

- Advanced Features

- Macro definitions

```
DEBUG = -g -DDEBUG
CFLAGS = -Wall -ansi $(DEBUG)
...
Program: Program.c Program.h
gcc $(CFLAGS) Program.c -o Program
```

- Dummy targets (aka. *phony* targets)

```
all: PhotoLab2

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

- More features...

- `man make`

EECS22: Advanced C Programming, Lecture 8

(c) 2011 R. Doemer

28

Application Example

- Advanced Makefile for PhotoLab2 (part 1/2)

```
# Makefile: PhotoLab2

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

# dummy 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
```