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

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

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

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

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

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**

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**

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"`**

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"`**

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);`

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);`

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)
  - executes necessary build commands automatically
- **`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

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!

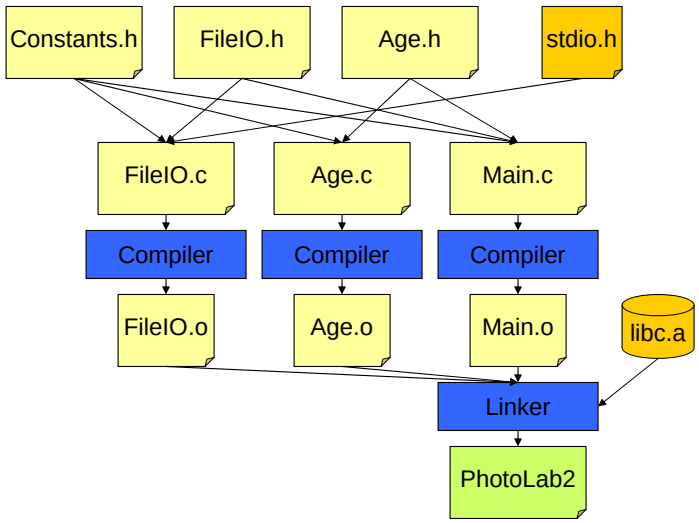
```
% vi HelloWorld.c
% vi Makefile
% make
gcc -Wall -ansi HelloWorld.c -o HelloWorld
% HelloWorld
Hello World!
%
```

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!

Application Example

- Making **`PhotoLab2`**



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
```

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) 2013 R. Doemer 29

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) 2013 R. Doemer 30

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
```