

EECS 22: Advanced C Programming

Lecture 9

Rainer Dömer

doemer@uci.edu

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

Lecture 9: Overview

- Warm-up Quiz
- Course Administration
 - Midterm course evaluation
- Assertions
 - Using and disabling assertions
- Debugging
 - Source-level debugger `gdb`
 - Running the program under debugger control
 - Navigating and inspecting the stack
 - Inspecting and modifying variable values
 - Advanced commands for using break points
 - Data display debugger `ddd`

Quiz: Question 11


- Today's computers run at which clock speed?
 - a) 85 MPH
 - b) 1 kHz
 - c) 1 ms
 - d) 1 GHz
 - e) 1 MHz

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

3

Quiz: Question 11

- Today's computers run at which clock speed?
 - a) 85 MPH
 - b) 1 kHz
 - c) 1 ms
 -  d) 1 GHz
 - e) 1 MHz

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

4

Quiz: Question 12

- Which of the following names are valid keywords in C? (Check all that apply!)
- a) `if`
 - b) `when`
 - c) `void`
 - d) `main`
 - e) `Int`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

5

Quiz: Question 12

- Which of the following names are valid keywords in C? (Check all that apply!)
- a) `if`
 - b) `when`
 - c) `void`
 - d) `main`
 - e) `Int`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

6

Quiz: Question 13

- Which of the following names are valid identifiers in C? (Check all that apply!)
- a) `xyz`
 - b) `IBM`
 - c) `dollar amount`
 - d) `My_Very_Long_Variable_Name`
 - e) `2fast4you`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

7

Quiz: Question 13

- Which of the following names are valid identifiers in C? (Check all that apply!)
- a) `xyz`
 - b) `IBM`
 - c) `dollar amount`
 - d) `My_Very_Long_Variable_Name`
 - e) `2fast4you`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

8

Quiz: Question 14



- Which of the following constructs are valid type names in C? (Check all that apply!)
 - a) `long char`
 - b) `long double`
 - c) `signed long long`
 - d) `unsigned float`
 - e) `signed integer`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

9

Quiz: Question 14

- Which of the following constructs are valid type names in C? (Check all that apply!)
 - a) `long char`
 -  b) `long double`
 -  c) `signed long long`
 - d) `unsigned float`
 - e) `signed integer`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

10

Quiz: Question 15




- Which of the following constants is of type **double**?
(Check all that apply!)
 - a) **42**
 - b) **.42**
 - c) **4e2**
 - d) **4E2**
 - e) **42f**

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

11

Quiz: Question 15

- Which of the following constants is of type **double**?
(Check all that apply!)
 - a) 42
 -  b) **.42**
 -  c) **4e2**
 -  d) **4E2**
 - e) 42f

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

12

Quiz: Question 16

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

```
x = 3 << 2 >> 1;
```

- a) **Syntax Error!**
- b) 3
- c) 6
- d) 12
- e) 321

EECS22: Advanced C Programming, Lecture 9


(c) 2012 R. Doemer

13

Quiz: Question 16

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

```
x = 3 << 2 >> 1;
```

- a) Syntax Error!
- b) 3
-  c) 6
- d) 12
- e) 321

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

14

Quiz: Question 17



- Which of the following expressions correctly computes the polynomial $p = 2x^2 - 3x + 4$?
(Check all that apply!)
 - `p = 2x^2 - 3x + 4;`
 - `p = 2xx - 3x + 4;`
 - `p = x*x*2 - 3*x + 4.0;`
 - `p = 2*(x*x + 3)*x + 4;`
 - `p = (2*x - 3)*x + 4;`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

15

Quiz: Question 17

- Which of the following expressions correctly computes the polynomial $p = 2x^2 - 3x + 4$?
(Check all that apply!)
 - `p = 2x^2 - 3x + 4;`
 - `p = 2xx - 3x + 4;`
 -  `p = x*x*2 - 3*x + 4.0;`
 - `p = 2*(x*x + 3)*x + 4;`
 -  `p = (2*x - 3)*x + 4;`

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

16

Quiz: Question 18

- What is the result of the evaluation of the following expression?


```
1 == 2 || 3 < 4 && 5 > 6
```

- a) 123456
- b) true
- c) false
- d) 1
- e) 0

Quiz: Question 18

- What is the result of the evaluation of the following expression?

```
1 == 2 || 3 < 4 && 5 > 6
```

- a) 123456
- b) true
- c) false
- d) 1
-  e) 0

Quiz: Question 19

- What is the result of the evaluation of the following expression?


```
17 < 42 ? 17 : 42
```

- a) 1742
- b) 17
- c) 42
- d) true
- e) false

Quiz: Question 19

- What is the result of the evaluation of the following expression?

```
17 < 42 ? 17 : 42
```

- a) 1742
-  b) 17
- c) 42
- d) true
- e) false

Quiz: Question 20

- For integer $x = 1$ at the beginning, what is the value of x after the following statement?


```
x += x + 1;
```

- a) 0
- b) 1
- c) 2
- d) 3
- e) 4

Quiz: Question 20

- For integer $x = 1$ at the beginning, what is the value of x after the following statement?

```
x += x + 1;
```

- a) 0
- b) 1
- c) 2
-  d) 3
- e) 4

Course Administration

- Midterm Course Evaluation
 - One week, starting this Sunday!
 - Sunday, Oct. 28, noon – Sunday, Nov. 4, noon
 - Online via EEE Evaluation application
- Feedback from students to instructors
 - Completely voluntary
 - Completely anonymous
 - Very valuable
 - Help to improve this class!
- Mandatory Final Course Evaluation
 - expected for week 10 (TBA)

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

23

Assertions

- Run-time Checks for Diagnostics and Debugging
 - Can be manually implemented

```
...
#ifdef DEBUG
if (value > 100)
{ printf("Value is over 100!");
  abort();
} /* fi */
#endif /* DEBUG */
...
```

- Assertions
 - Diagnostics provided by standard C library

```
...
#include <assert.h>
...
assert(value <= 100);
...
```

EECS22: Advanced C Programming, Lecture 9

(c) 2012 R. Doemer

24

Assertions

- Diagnostics provided by standard C library

```
#include <assert.h>
...
assert(value <= 100);
```

- Header file `assert.h`
 - Defines `assert(condition)` as a preprocessor macro
- Assertion failure
 - At run-time, if `condition` evaluates to `false`, the program is aborted with a diagnostic message

```
assertion: program.c:12: main: Assertion `value <= 100' failed.
Abort
```

- Disabling assertions
 - If `NDEBUG` is defined when `assert.h` is included, the `assert()` macro is ignored (empty)
 - Define `NDEBUG` for final programs delivered to the end user!

Assertions

- Recommendations on Using Assertions

- Use assertions often
 - Confirm assumptions about parameters, calculated values, etc.
 - Assertions are cheap (low run-time overhead)!
- Use assertions from beginning during software development
 - Diagnostic messages are very helpful in development
 - Program aborts as soon as a value is out of expected range
 - Location and problem situation are shown
 - This can avoid more serious problems later
- Disable assertions for final program
 - Diagnostic messages are of no use to the end user!
 - User has no idea about condition and source location
- Beware of side-effects in assertions
 - Implemented as a macro!
 - Can lead to *Heisenbugs* which disappear when debugging is on!

Debugging

- Source-level Debugger `gdb`
 - Debugging features
 - run the program under debugger control
 - follow the control flow of the program during execution
 - set breakpoints to stop execution at specific points
 - inspect (and adjust) the values of variables
 - find the point in the program where the “crash” happens
 - Preparation:
 - compile your program with debugging support on
 - Option `-g` tells compiler to add debugging information (symbol tables) to the generated executable file
 - `gcc -g Program.c -o Program -Wall -ansi`
 - `gdb Program`

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

27

Debugging

- Source-level Debugger `gdb`
 - Running the program under debugger control
 - `run`
 - starts the execution of the program in the debugger
 - `break function_name (or file:line_number)`
 - inserts a breakpoint; program execution will stop at the breakpoint
 - `cont`
 - continues the execution of the program in the debugger
 - `list from_line_number, to_line_number`
 - lists the current or specified range of line_numbers
 - `print variable_name`
 - prints the current value of the variable `variable_name`
 - `next`
 - executes the next statement (one statement at a time)
 - `quit`
 - exits the debugger (and terminates the program)
 - `help`
 - provides helpful details on debugger commands

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

28

Debugging

- Example session: `Cylinder.c`

```
% vi Cylinder.c
% gcc Cylinder.c -Wall -ansi -o Cylinder -g
% gdb Cylinder
GNU gdb (GDB) Red Hat Enterprise Linux (7.0.1-37.el5_7.1)
Copyright (C) 2009 Free Software Foundation, Inc.
...
Reading symbols from
/users/faculty/doemer/eecs22/lecture10/Cylinder...done.
(gdb) break main
Breakpoint 1 at 0x400654: file Cylinder.c, line 48.
(gdb) run
Starting program: /users/faculty/doemer/eecs22/lecture10/Cylinder
Breakpoint 1, main () at Cylinder.c:48
48     printf("Please enter the radius!\n");
(gdb) next
Please enter the radius!
49     scanf("%lf", &r);
...
```

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

29

Debugging

- Example session: `Cylinder.c`

```
...
(gdb) next
5
50     printf("Please enter the height!\n");
(gdb) print r
$1 = 5
(gdb) cont
Continuing.
Please enter the height!
10
The surface area is 471.238905.
The volume is 785.398175.
Program exited normally.
(gdb) quit
%
```

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

30

Debugging

- Source-level Debugger `gdb` (continued)
 - Navigating the stack
 - `step`
 - steps into a function call
 - `finish`
 - continues execution until the current function has returned
 - `where`
 - shows where in the function call hierarchy you are
 - prints a *back trace* of current *stack frames*
 - `up`
 - steps up one stack frame (up into the caller)
 - `down`
 - steps down one stack frame (down into the callee)

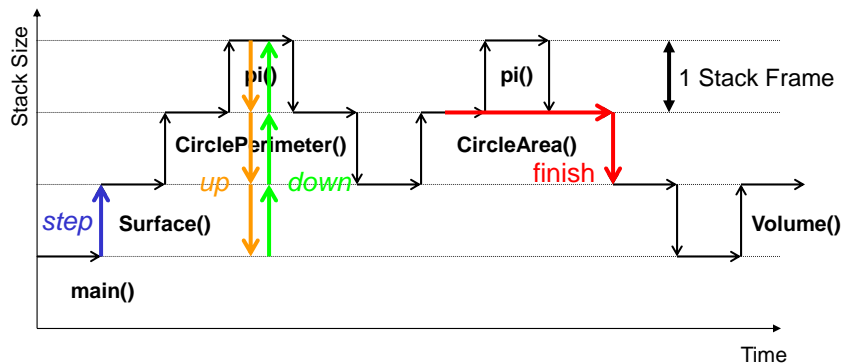
EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

31

Debugging

- Navigating Stack Frames in the Debugger
 - `step`: execute and step into a function call
 - `up`, `down`: navigate stack frames
 - `finish`: resume execution until the end of the current function



EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

32

Debugging

- Example session: `Cylinder.c`

```
% vi Cylinder.c
% gcc Cylinder.c -o Cylinder -Wall -ansi -g
% gdb Cylinder
GNU gdb 6.3
(gdb) break 55
Breakpoint 1 at 0x108d0: file Cylinder.c, line 55.
(gdb) run
Starting program: /users/faculty/doemer/eecs10/Cylinder/Cylinder
Please enter the radius: 10
Please enter the height: 10
Breakpoint 1, main () at Cylinder.c:56
56      s = Surface(r, h);
(gdb) step
Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
(gdb) step
CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
...
EE
```

Debugging

- Example session: `Cylinder.c`

```
(gdb) step
pi () at Cylinder.c:14
14      return(3.1415927);
(gdb) where
#0  pi () at Cylinder.c:14
#1  0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
#2  0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
#3  0x000108e0 in main () at Cylinder.c:56
(gdb) up
#1  0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
(gdb) up
#2  0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
(gdb) up
#3  0x000108e0 in main () at Cylinder.c:56
56      s = Surface(r, h);
...
EE
```

Debugging

- Example session: `Cylinder.c`

```
(gdb) down
#2 0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
(gdb) down
#1 0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
(gdb) down
#0 pi () at Cylinder.c:14
14      return(3.1415927);
(gdb) finish
Run till exit from #0 pi () at Cylinder.c:14
0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
Value returned is $1 = 3.1415926999999999
(gdb) finish
Run till exit from #0 CirclePerimeter (r=10) at Cylinder.c:24
0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
...
EE
```

Debugging

- Example session: `Cylinder.c`

```
Value returned is $2 = 62.831854
(gdb) next
32      lid = CircleArea(r);
(gdb) step
CircleArea (r=10) at Cylinder.c:19
19      return(pi() * r * r);
(gdb) finish
Run till exit from #0 CircleArea (r=10) at Cylinder.c:19
0x00010818 in Surface (r=10, h=10) at Cylinder.c:32
32      lid = CircleArea(r);
Value returned is $3 = 314.15926999999999
(gdb) cont
Continuing.
The surface area is 1256.637080.
The volume is 3141.592700.
Program exited normally.
(gdb) quit
%
```

Debugging

- Source-level Debugger `gdb` (continued)
 - Inspecting the stack
 - `info frame`
 - displays information about the current stack frame
 - `info locals`
 - lists the local variables in the current function (current stack frame)
 - `info scope function`
 - lists the variables in the scope of the specified function
 - Calling functions (outside of the regular control flow)
 - `call function(arguments)`
 - calls the specified function with the specified arguments
 - Assembly level inspection
 - `info registers`
 - lists the CPU registers and their contents
 - `disassemble function`
 - disassembles the function and lists its assembly code

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

37

Debugging

- Source-level Debugger `gdb` (continued)
 - Inspecting and modifying variable values
 - `print variable_name`
 - prints the current value of the variable `variable_name`
 - `set variable = value`
 - sets the specified variable to the specified value
 - `display variable`
 - prints the value of a variable each time before the next command
 - `info display`
 - lists information on the displayed variables
 - `undisplay variable`
 - turns off the display of the specified variable

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

38

Debugging

- Source-level Debugger `gdb` (continued)
 - Advanced commands for using break points
 - `info breakpoints`
 - displays information about break points
 - `tbreak function_name (Or file:line_number)`
 - inserts a temporary breakpoint (valid only once)
 - `watch variable`
 - sets a watch point on the specified variable for write access
 - `rwatch variable`
 - sets a watch point on the specified variable for read access
 - `ignore breakpoint n`
 - skips the specified break point n times
 - `enable (Or disable) breakpoint (Or watchpoint)`
 - Enables (or disables) a break point (or watch point)
 - `condition breakpoint condition`
 - Specifies a condition for the given break point

EECS22: Advanced C Programming, Lecture 10

(c) 2012 R. Doemer

39

Debugging

- Data Display Debugger `ddd`
 - Graphical frontend for `gdb`
 - Requires `X` forwarding and supportive client (e.g. `Xming` in addition to `Putty`)
 - Provides menu bar and command buttons
 - Displays separate work windows
 - Graphical display area for data structures
 - Source code browser
 - Assembly code browser
 - Command line interface
 - Example: `Cylinder.c`

```

000: /users/faculty/doemer/eeecs22/lecture9/Cylinder.c
File Edit View Program Commands Status Source Data Help
0 Cylinder.c:55
Run| Interrupt| Step| Next| Next| Finish| Cont| Kill| Up| Down| Undo| Redo| Edit| Make
1: r 2: n 3: s
10 20 1884.95562
48 printf("Please enter the radius\n");
49 scanf("%f", &r);
50 printf("Please enter the height\n");
51 scanf("%f", &h);
52
53 /* computation section */
54 s = Surface(r, h);
55 v = Volume(r, h);
56
57 /* output section */
58 printf("The surface area is %f.\n", s);

(gdb) graph display s
(gdb) break Cylinder.c:55
Breakpoint 2 at 0x4000ba: file Cylinder.c, line 55.
(gdb) break Cylinder.c:55
Breakpoint 3 at 0x4000ba: file Cylinder.c, line 55.
(gdb) break Cylinder.c:55
Breakpoint 4 at 0x4000ba: file Cylinder.c, line 55.
(gdb) clear Cylinder.c:55
Deleted breakpoints 2 3 4
(gdb) ]
Deleted breakpoints 2 3 4
  
```

EECS22: Advanced C Programming, Lecture 17

(c) 2011 R. Doemer

40