

EECS 22: Advanced C Programming

Lecture 10

Rainer Dömer

doemer@uci.edu

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

Lecture 10: Overview

- Course Administration
 - Midterm course evaluation
- Warm-up Quiz
- 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

Course Administration

- Midterm Course Evaluation
 - One week, starting today!
 - Tuesday, Oct. 25, 11am – Monday, Oct. 31, 11pm
 - 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) 2011 R. Doemer

3

Quiz: Question 21

- Which of the following variable declarations is valid in ANSI-C?
(Check all that apply! 2 pts.)
 - `double xyz;`
 - `double x, y, z;`
 - `double x = 1.0;`
 - `double x = 1.1, y = 2.2, z = 3.3;`
 - `double x,y,z = 1.0,2.0,3.0;`

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

4

Quiz: Question 21

- Which of the following variable declarations is valid in ANSI-C?

(Check all that apply! 2 pts.)


- a) `double xyz;`
- b) `double x, y, z;`
- c) `double x = 1.0;`
- d) `double x = 1.1, y = 2.2, z = 3.3;`
- e) `double x,y,z = 1.0,2.0,3.0;`

Quiz: Question 22

- Which of the following data types has the largest range of representable numbers?

- a) `char`
- b) `short int`
- c) `long long int`
- d) `unsigned int`
- e) `signed long int`

Quiz: Question 22

- Which of the following data types has the largest range of representable numbers?
 - a) `char`
 - b) `short int`
 -  c) `long long int`
 - d) `unsigned int`
 - e) `signed long int`

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

7

Quiz: Question 23

- Which of the following data types can store the greatest value?
 - a) `long int`
 - b) `long long int`
 - c) `unsigned long long int`
 - d) `float`
 - e) `double`

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

8

Quiz: Question 23

- Which of the following data types can store the greatest value?
 - a) `long int`
 - b) `long long int`
 - c) `unsigned long long int`
 - d) `float`
 - e) **`double`**

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

9

Quiz: Question 24

- Assuming that `x` is a variable of type `int`, which values of `x` satisfy the following condition?

```
x % 2 == 1
```

- a) no value
- b) any value
- c) any value less than 2
- d) any odd value
- e) any even value

EECS22: Advanced C Programming, Lecture 10


(c) 2011 R. Doemer

10

Quiz: Question 24

- Assuming that `x` is a variable of type `int`, which values of `x` satisfy the following condition?

```
x % 2 == 1
```

- a) no value
- b) any value
- c) any value less than 2
-  d) any odd value
- e) any even value

Quiz: Question 25

- Assume that `x` is an integer in the range of 1 through 10 inclusively. Which of the following expressions can be used as a test for `x` being an even number?

(Check all that apply! 2 pts.)

- a) `x % 2 == 0`
- b) `x / 2 > 1`
- c) `x % 2 == 1`
- d) `x / 2 * 2 == x`
- e) `x==2 || x==4 || x==6 || x==8 || x==10`

Quiz: Question 25

- Assume that x is an integer in the range of 1 through 10 inclusively. Which of the following expressions can be used as a test for x being an even number?

(Check all that apply! 2 pts.)

- a) $x \% 2 == 0$
- b) $x / 2 > 1$
- c) $x \% 2 == 1$
- d) $x / 2 * 2 == x$
- e) $x==2 \ || \ x==4 \ || \ x==6 \ || \ x==8 \ || \ x==10$

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

13

Quiz: Question 26

- Given the following function g , what is the result of $g(85)$?

- a) `'A'`
- b) `'B'`
- c) `'C'`
- d) `'D'`
- e) `'F'`

```
char g(int n)
{
    switch(n/10)
    { case 10:
      case 9: return('A');
      case 8: return('B');
      case 7: return('C');
      case 6: return('D');
      default: return('F');
    }
}
```


EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

14

Quiz: Question 26

- Given the following function `g`, what is the result of `g(85)`?

- a) 'A'
-  b) 'B'
- c) 'C'
- d) 'D'
- e) 'F'

```
char g(int n)
{
    switch(n/10)
    { case 10:
      case 9: return('A');
      case 8: return('B');
      case 7: return('C');
      case 6: return('D');
      default: return('F');
    }
}
```

Quiz: Question 27

- What is the value of `x` after the following code fragment is executed?


```
int x = 0;
for(x = 1; x <= 10; x++)
{ }
```

- a) 0
- b) 1
- c) 9
- d) 10
- e) 11

Quiz: Question 27

- What is the value of `x` after the following code fragment is executed?

```
int x = 0;
for(x = 1; x <= 10; x++)
{ }
```

- a) 0
- b) 1
- c) 9
- d) 10
-  e) 11

Quiz: Question 28

- Given the following program fragment, what is printed when it gets executed?

```
int i = 1;
int s = 0;
while (1)
{ i++;
  if (i >= 10)
  { break; }
  if (i % 2 == 1)
  { continue; }
  s += i;
}
printf("%d", s);
```

- a) nothing
- b) 0
- c) 10
- d) 20
- e) 30

Quiz: Question 28

- Given the following program fragment, what is printed when it gets executed?

- a) nothing
- b) 0
- c) 10
- d) 20
- e) 30

```
int i = 1;
int s = 0;
while (1)
{
    i++;
    if (i >= 10)
        { break; }
    if (i % 2 == 1)
        { continue; }
    s += i;
}
printf("%d", s);
```

Quiz: Question 29

- Given the following code fragment, which of the following statements are true?
(Check all that apply!)

- a) Function `f` is declared.
- b) Function `g` calls function `f`
- c) Variable `z` is a local variable of function `g`
- d) Function `g` is declared and defined.
- e) `y` is a parameter of function `g`.

```
double f(int x);
void g(int x, int y)
{
    int z;

    z = f(x) + 2*y;
    return z;
}
```

Quiz: Question 29

- Given the following code fragment, which of the following statements are true?
(Check all that apply!)

```
double f(int x);
void g(int x, int y)
{
    int z;

    z = f(x) + 2*y;
    return z;
}
```

- a) Function `f` is declared.
- b) Function `g` calls function `f`
- c) Variable `z` is a local variable of function `g`
- d) Function `g` is declared and defined.
- e) `y` is a parameter of function `g`.

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

21

Quiz: Question 30

- Given the following program fragment, what is the value of `g(2, f(3, 4))`?

```
int x = 7;

int f(int x, int y)
{
    return x + y;
}

int g(int x, int y)
{
    return f(y, x);
}
```

- a) 8
- b) 9
- c) 10
- d) 11
- e) 12


EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

22

Quiz: Question 30

- Given the following program fragment, what is the value of `g(2, f(3, 4))`?

- a) 8
-  b) 9
- c) 10
- d) 11
- e) 12

```
int x = 7;

int f(int x, int y)
{
    return x + y;
}

int g(int x, int y)
{
    return f(y, x);
}
```

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

23

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) 2011 R. Doemer

24

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) 2011 R. Doemer

25

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.e15_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) 2011 R. Doemer

26

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
%
```

Debugging

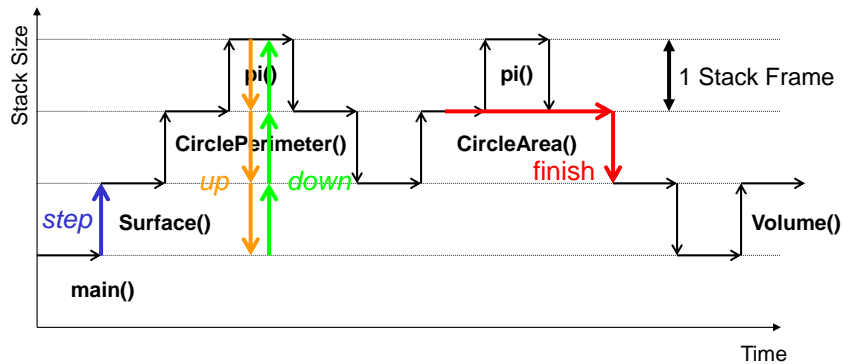
- Source-level Debugger `gdb` (continued)

- 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)

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) 2011 R. Doemer 29

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);
...

```

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);
...
```

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

31

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
%
```

EECS22: Advanced C Programming, Lecture 10

(c) 2011 R. Doemer

33

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) 2011 R. Doemer

34

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

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