EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 14

Rainer Dömer

doemer@uci.edu

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

Lecture 14: Overview

- Course administration
  - Reminder: Midterm course evaluation
- Functions
  - Terms and concepts
  - Scope rules
  - Scope example
- Debugging
  - Stack frames

Course Administration

- Midterm Course Evaluation
  - Open until Monday, 8pm!
  - Oct. 29, 2008, 8am - Nov. 3, 2008, 8pm
  - 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)

Functions

- Review: Terms and Concepts
  - Function declaration
    - function prototype with name, parameters, and return type
  - Function parameters
    - formal parameters holding the data supplied to a function
  - Function definition
    - extended declaration, defines the behavior in function body
  - Local variables
    - variables defined locally in a function body
  - Function call
    - expression invoking a function with supplied arguments
  - Function arguments
    - arguments passed to a function call (initial values for parameters)
  - Return value
    - result computed by a function call

Functions

- *Scope* of an identifier
  - Portion of the program where the identifier can be referenced
  - aka. accessability, visibility
- Scope rules
  - Global variables: *file scope*
    - Declaration outside any function (at global level)
    - Scope in entire source file after declaration
  - Function parameters: *function scope*
    - Declaration in function parameter list
    - Scope limited to this function body (entirely)
  - Local variables: *block scope*
    - Declaration inside a compound statement (i.e. function body)
    - Scope limited to this compound statement block (entirely)

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);

int x = 5,
    y = 7;

int square(int a)
{  int s;
   s = a * a;
   return s;
}

int add_y(int x)
{  int s;
   s = x + y;
   return s;
}

int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Header file inclusion

Function declarations

Global variables

Function definition
Local variable

Function definition
Local variable

Function definition
Local variable

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of global functions **`printf()`**, **`scanf()`**, etc.

EECS10: Computational Methods in ECE, Lecture 14 (c) 2008 R. Doemer 7

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of global function **`square()`**

EECS10: Computational Methods in ECE, Lecture 14 (c) 2008 R. Doemer 8

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);

int x = 5,
    y = 7;

int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of global function **`add_y()`**

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);

int x = 5,
    y = 7;

int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of global variable **`x`**

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of global variable **y**

EECS10: Computational Methods in ECE, Lecture 14 (c) 2008 R. Doemer 11

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of parameter **a**

EECS10: Computational Methods in ECE, Lecture 14 (c) 2008 R. Doemer 12

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of local variable **s**

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Local variables are independent!
(unless their scopes are nested)

Scope of local variable **s**

Scope of local variable **s**

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Local variables are independent!
(unless their scopes are nested)

Scope of local variable **s**

Scope of local variable **s**

Scope of local variable **z**

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Scope of parameter **x**

Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{  int s;
   s = a * a;
   return s;
}
int add_y(int x)
{  int s;
   s = x + y;
   return s;
}
int main(void)
{  int z;
   z = square(x);
   z = add_y(z);
   printf("%d\n", z);
   return 0;
}
```

Shadowing!
In nested scopes,
inner scope takes precedence!

Scope of global variable **x**

Scope of parameter **x**

Debugging

- Source-level Debugger **gdb**
  - Basic **gdb** commands
    - **run**
      - starts the execution of the program in the debugger
    - **break *function_name***
      - inserts a breakpoint at ***function_name***
      - program execution will stop at the breakpoint
    - **list *line_numbers***
      - lists the current or specified ***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

Debugging

- Source-level Debugger **gdb** (continued)
  - Additional **gdb** commands
    - **step**
      - steps into a function call
    - **finish**
      - continues execution until the current function is finished
    - **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)
    - **info locals**
      - lists the local variables in the current function (current stack frame)
    - **info scope *function_name***
      - lists the variables in scope of the ***function_name***

Scope Rules: Example

- Program example: **Scope.c** (part 1/2)

```
/* Scope.c: example demonstrating scope rules   */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 10/30/04 RD  initial version                 */

#include <stdio.h>

int square(int a);      /* global function declarations */
int add_y(int x);

int x = 5,              /* global variables */
    y = 7;

int square(int a)       /* global function definition */
{
   int s;               /* local variable */

   s = a * a;
   return s;
}
...
```

Scope Rules: Example

- Program example: **`Scope.c`** (part 2/2)

```
...
int add_y(int x)        /* global function definition */
{
   int s;               /* local variable */

   s = x + y;
   return s;
}
int main(void)          /* main function definition */
{
   int z;               /* local variable */

   z = square(x);
   z = add_y(z);

   printf("%d, %d, %d\n", x, y, z);
   return 0;
}
/* EOF */
```


Scope Rules: Example

- Example session: **`Scope.c`** (part 1/3)

```
% vi Scope.c
% gcc Scope.c -o Scope -Wall –ansi -g
% Scope
5, 7, 32
% gdb Scope
GNU gdb 5.0
[...]
(gdb) break main
Breakpoint 1 at 0x1079c: file Scope.c, line 36.
(gdb) run
Starting program: /users/faculty/doemer/eecs10/Scope/Scope

Breakpoint 1, main () at Scope.c:36
36          z = square(x);
(gdb) step
square (a=5) at Scope.c:20
20          s = a * a;
(gdb) next
21          return s;
...
```

Scope Rules: Example

- Example session: **`Scope.c`** (part 2/3)

```
...
(gdb) next
22      }
(gdb) next
main () at Scope.c:37
37          z = add_y(z);
(gdb) step
add_y (x=25) at Scope.c:28
28          s = x + y;
(gdb) where
#0  add_y (x=25) at Scope.c:28
#1  0x107c4 in main () at Scope.c:37
(gdb) up
#1  0x107c4 in main () at Scope.c:37
37          z = add_y(z);
(gdb) down
#0  add_y (x=25) at Scope.c:28
28          s = x + y;
...
```


Scope Rules: Example

- Example session: **`Scope.c`** (part 3/3)

```
...
(gdb) finish
Run till exit from #0  add_y (x=25) at Scope.c:28
0x107c4 in main () at Scope.c:37
37          z = add_y(z);
Value returned is $1 = 32
(gdb) info locals
z = 25
(gdb) info scope square
Scope for square:
Symbol a is an argument at stack/frame offset 68, length 4.
Symbol s is a local variable at frame offset -20, length 4.
(gdb) info scope add_y
Scope for add_y:
Symbol x is an argument at stack/frame offset 68, length 4.
Symbol s is a local variable at frame offset -20, length 4.
(gdb) quit
%
```