

Lecture 6.2: Overview

- Functions
 - Terms and concepts
 - Scope rules
 - Scope example
- Debugging
 - Scopes

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. accessibility, 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)

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

3

Scope Rules: Example

| | |
|--|---------------------------------------|
| <code>#include <stdio.h></code> | Header file inclusion |
| <code>int square(int a);</code> <code>int add_y(int x);</code> | Function declarations |
| <code>int x = 5,</code> <code> y = 7;</code> | Global variables |
| <code>int square(int a)</code> <code>{ int s;</code> <code> s = a * a;</code> <code> return s;</code> <code>}</code> | Function definition Local variable |
| <code>int add_y(int x)</code> <code>{ int s;</code> <code> s = x + y;</code> <code> return s;</code> <code>}</code> | Function definition Local variable |
| <code>int main(void)</code> <code>{ int z;</code> <code> z = square(x);</code> <code> z = add_y(z);</code> <code> printf("%d\n", z);</code> <code> return 0;</code> <code>}</code> | Function definition Local variable |

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

4

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 6

(c) 2013 R. Doemer

5

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 6

(c) 2013 R. Doemer

6

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()`

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 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 variable
x

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 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 variable
y

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

9

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 6

(c) 2013 R. Doemer

10

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

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 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;
}
```

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

Scope of local variable
s

Scope of local variable
s

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 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;
}
```

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

Scope of local variable

s

Scope of local variable

s

Scope of local variable

z

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

13

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

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

14

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

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

15

Debugging

- Source-level Debugger `gdb`
 - Basic `gdb` commands
 - `run`
 - starts the execution of the program in the debugger
 - `break function_name (or 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

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

16

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*

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

17

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;
}
...

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

18

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 */

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

19

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;
...

```

EE

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;
...

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

21

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
%

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

22