

# EECS 10: Computational Methods in Electrical and Computer Engineering

## 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

- Functions
  - Introduction to function concepts
    - Function declaration
    - Function definition
    - Function call
  - Simple functions
    - Example `square.c`
  - Hierarchy of functions
    - Example `cylinder.c`
  - Function call graph
  - Function call trace
  - Function call stack
- Debugging
  - Navigating stack frames

## Functions

- Introduction to Functions
  - Important programming concepts
    - Hierarchy
    - Encapsulation
    - Information hiding
    - Divide and conquer
  - Software reuse
    - Don't re-invent the wheel!
  - Program composition
    - C program = Set of functions
      - starting point: function named `main`
    - Libraries = Set of functions
      - predefined functions (typically written by somebody else)

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

3

## Functions

- C programming language distinguishes 3 constructs around functions
  - *Function declaration*
    - declaration of function name, parameters, and return type
  - *Function definition*
    - extension of a function declaration with a function body
    - definition of the function behavior
  - *Function call*
    - invocation of a function

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

4

## Functions

- Function Declaration
  - aka. *function prototype* or *function signature*
  - declares
    - function name
    - function parameters
    - type of return value
- Example:

```
double Square(double p);
```

  - function is named **square**
  - function takes one parameter **p** of type **double**
  - function returns a value of type **double**

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

5

## Functions

- Function Definition
  - extends a function declaration with a function body
  - defines the statements executed by the function
  - may use local variables for the computation
  - returns result value via **return** statement (if any)
- Example:

```
double Square(double p)
{
    double r;
    r = p * p;
    return r;
}
```

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

6

## Functions

- Function Call
  - expression invoking a function
  - supplies arguments for formal parameters
  - invokes the function
  - result is the value returned by the function
- Example:
 

```
double a, b;
b = square(a);
```

  - function `square` is called
  - argument `a` is passed for parameter `p` (by value)
  - value returned by the function is assigned to `b`

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

7

## Functions

- C programming language distinguishes 3 constructs
  - Function declaration
    - declaration of function name, parameters, and return type
  - Function definition
    - extension of a function declaration with a function body
    - definition of the function behavior
  - Function call
    - invocation of a function
- C program rules
  - A function must be declared before it can be called.
  - Multiple function declarations are allowed (if they match).
  - A function definition is an implicit function declaration.
  - A function must be defined exactly once in a program.
  - A function may be called any number of times.

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

8

## Functions

- Program example: `square.c` (part 1/2)

```

/* Square.c: example demonstrating functions */
/* author: Rainer Doemer */
/* modifications: */
/* 10/27/08 RD renamed parameters and arguments */
/* 10/27/04 RD initial version */

#include <stdio.h>

/* function declaration */
double square(double p);

/* function definition */
double square(double p)
{
    double r;
    r = p * p;
    return r;
} /* end of square */

...

```

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

9

## Functions

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

```

...
/* main function */
int main(void)
{
    /* variable definitions */
    double a, b;

    /* input section */
    printf("Please enter a value for the argument: ");
    scanf("%lf", &a);

    /* computation section */
    b = square(a);

    /* output section */
    printf("The square of %g is %g.\n", a, b);

    /* exit */
    return 0;
} /* end of main */

/* EOF */

```

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

10

## Functions

- Example session: `square.c`

```
% vi square.c
% gcc square.c -o square -Wall -ansi
% ./square
Please enter a value for the argument: 3
The square of 3 is 9.
% ./square
Please enter a value for the argument: 5.5
The square of 5.5 is 30.25.
%
```

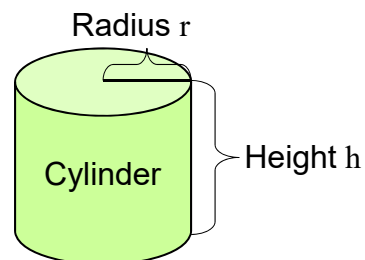
## Functions

- Hierarchy of Functions
  - functions call other functions

- Example:

### Cylinder calculations

- given radius and height
- calculate surface and volume



- Circle constant  $\pi = 3.14159265\dots$
- Circle perimeter  $f_p(r) = 2 \times \pi \times r$
- Circle area  $f_a(r) = \pi \times r^2$
- Cylinder surface  $f_s(r, h) = f_p(r) \times h + 2 \times f_a(r)$
- Cylinder volume  $f_v(r, h) = f_a(r) \times h$

## Functions

- Program example: `Cylinder.c` (part 1/3)

```
/* Cylinder.c: cylinder functions */
/* author: Rainer Doemer          */
/* modifications:                 */
/* 10/25/05 RD initial version    */

#include <stdio.h>

/* cylinder functions */

double pi(void)
{
    return(3.1415927);
}

double CircleArea(double r)
{
    return(pi() * r * r);
}
...
```

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

13

## Functions

- Program example: `Cylinder.c` (part 2/3)

```
...
double CirclePerimeter(double r)
{
    return(2 * pi() * r);
}

double Surface(double r, double h)
{
    double side, lid;
    side = CirclePerimeter(r) * h;
    lid = CircleArea(r);
    return(side + 2*lid);
}

double Volume(double r, double h)
{
    return(CircleArea(r) * h);
}
...
```

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

14

## Functions

- Program example: `Cylinder.c` (part 3/3)

```
...
/* main function */
int main(void)
{
    double r, h, s, v;

    /* input section */
    printf("Please enter the radius: ");
    scanf("%lf", &r);
    printf("Please enter the height: ");
    scanf("%lf", &h);

    /* computation section */
    s = Surface(r, h);
    v = Volume(r, h);

    /* output section */
    printf("The surface area is %f.\n", s);
    printf("The volume is %f.\n", v);

    return 0;
} /* end of main */
```

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

15

## Functions

- Example session: `Cylinder.c`

```
% vi Cylinder.c
% gcc Cylinder.c -o Cylinder -Wall -ansi
% ./Cylinder
Please enter the radius: 5.0
Please enter the height: 8.0
The surface area is 408.407051.
The volume is 628.318540.
%
```

EECS10: Computational Methods in ECE, Lecture 9

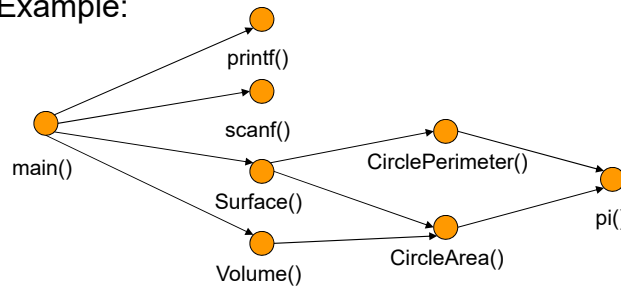
(c) 2019 R. Doemer

16



## Function Call Graph

- Graphical representation of function calls
  - Directed Graph
    - Vertices: Functions
    - Edges: Function calls
  - Shows dependencies among functions
  - Example:



EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

17

## Function Call Trace

- Sequence of function calls
  - Shows execution order of functions at run-time
- Example:

```

> main()
  > printf()
  > scanf()
  > printf()
  > scanf()
  > Surface()
    > CirclePerimeter()
      > pi()
    > CircleArea()
      > pi()
  > Volume()
    > CircleArea()
      > pi()
  > printf()
  > printf()
  
```

EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

18

## Function Call Stack

- Stack Frames
  - Keep track of active function calls
    - Stack grows by one frame with each function call
    - Stack shrinks by one frame with each completed function

The diagram illustrates the function call stack over time. The vertical axis represents 'Stack Size' and the horizontal axis represents 'Time'. The stack starts with a 'main()' frame. It then grows to include 'Surface()', then 'CirclePerimeter()', then 'pi()', then 'CircleArea()', then another 'pi()', and finally 'Volume()'. As each function completes, its frame is removed from the stack. A vertical double-headed arrow on the right side of the stack indicates a change of 1 Stack Frame.

EECS10: Computational Methods in ECE, Lecture 9 (c) 2019 R. Doemer 19

## Function Call Stack

- Stack Frames
  - Keep track of active function calls
    - Stack grows by one frame with each function call
    - Stack shrinks by one frame with each completed function

The diagram illustrates the function call stack over time. The vertical axis represents 'Stack Size' and the horizontal axis represents 'Time'. The stack starts with a 'main()' frame. It then grows to include 'Surface()', then 'CirclePerimeter()', then 'pi()', then 'CircleArea()', then another 'pi()', and finally 'Volume()'. As each function completes, its frame is removed from the stack. A vertical double-headed arrow on the right side of the stack indicates a change of 1 Stack Frame.

EECS10: Computational Methods in ECE, Lecture 9 (c) 2019 R. Doemer 20

## Function Call Stack

- Stack Frames
  - Keep track of active function calls
    - Stack grows by one frame with each function call
    - Stack shrinks by one frame with each completed function

Time

EECS10: Computational Methods in ECE, Lecture 9 (c) 2019 R. Doemer 21

## 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 9 (c) 2019 R. Doemer 22

## 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)

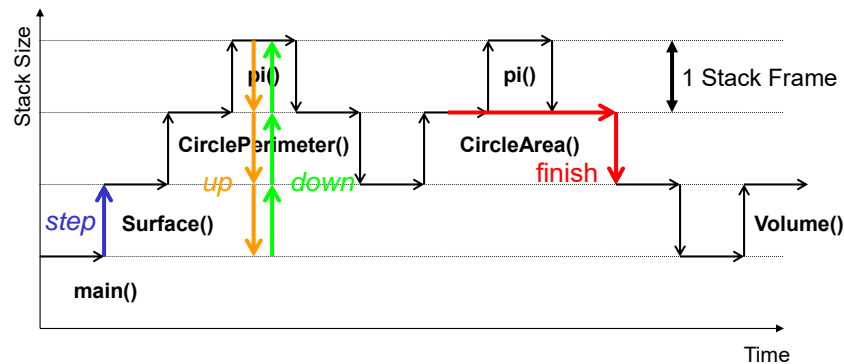
EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

23

## 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



EECS10: Computational Methods in ECE, Lecture 9

(c) 2019 R. Doemer

24

## 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
%
```