

EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 6

Rainer Dömer

doemer@uci.edu

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

Lecture 6.1: Overview

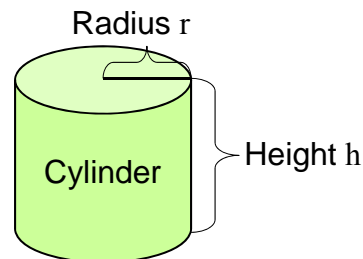
- Functions
 - Hierarchy of functions
 - Example `cylinder.c`
 - Function call graph
 - Function call trace
 - Function call stack
- Debugging
 - Navigating stack frames

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 $f_v(r, h) = f_a(r) \times h$

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

3

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 6

(c) 2013 R. Doemer

4

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 6

(c) 2013 R. Doemer

5

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 */

```

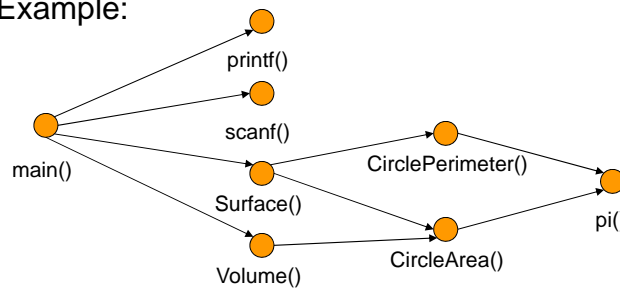
EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

6

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 6

(c) 2013 R. Doemer

7

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 6

(c) 2013 R. Doemer

8

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 6 (c) 2013 R. Doemer 9

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 6 (c) 2013 R. Doemer 10

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 6 (c) 2013 R. Doemer 11

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 12

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)

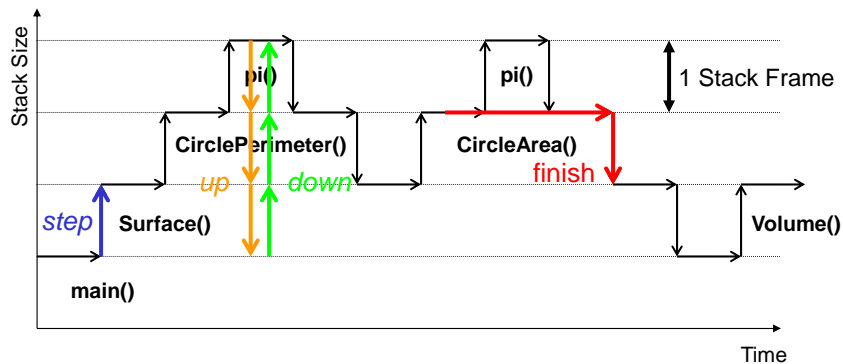
EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

13

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 6

(c) 2013 R. Doemer

14

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
%
```

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

21

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

22

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

23

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

24

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

25

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

26

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

27

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

28

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

29

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

30

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

31

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

32

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

33

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

34

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

35

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

36

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

37

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

39

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

40

Lecture 6.3: Overview

- Functions
 - Math library functions
 - Example `Function.c`
 - Standard library functions
 - Example `Dice.c`

Math Library Functions

- C standard math library
 - standard library supplied with every C compiler
 - predefined mathematical functions
 - e.g. $\cos(x)$, \sqrt{x} , etc.
- Math library header file
 - contains math function declarations
 - `#include <math.h>`
- Math library linker file
 - contains math function definitions (pre-compiled)
 - library file `libm.a`
 - compiler needs to *link* against the math library
 - use option `-llibraryname`
 - Example: `gcc MathProgram.c -o MathProgram -lm`

Math Library Functions

- Functions declared in `math.h` (part 1/2)

- <code>double sqrt(double x);</code>	\sqrt{x}
- <code>double pow(double x, double y);</code>	x^y
- <code>double exp(double x);</code>	e^x
- <code>double log(double x);</code>	$\log(x)$
- <code>double log10(double x);</code>	$\log_{10}(x)$
- <code>double ceil(double x);</code>	$\lceil x \rceil$
- <code>double floor(double x);</code>	$\lfloor x \rfloor$
- <code>double fabs(double x);</code>	$ x $
- <code>double fmod(double x, double y);</code>	$x \bmod y$

Math Library Functions

- Functions declared in `math.h` (part 2/2)

- <code>double cos(double x);</code>	$\cos(x)$
- <code>double sin(double x);</code>	$\sin(x)$
- <code>double tan(double x);</code>	$\tan(x)$
- <code>double acos(double x);</code>	$\arccos(x)$
- <code>double asin(double x);</code>	$\arcsin(x)$
- <code>double atan(double x);</code>	$\arctan(x)$
- <code>double cosh(double x);</code>	$\cosh(x)$
- <code>double sinh(double x);</code>	$\sinh(x)$
- <code>double tanh(double x);</code>	$\tanh(x)$

Math Library Functions

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

```

/* Function.c: compute a math function table */
/* */
/* author: Rainer Doemer */
/* */
/* modifications: */
/* 10/28/04 RD initial version */

#include <stdio.h>
#include <math.h>

/* function definition */

double f(double x)
{
    return cos(x);
} /* end of f */

...

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

45

Math Library Functions

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

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    double hi, lo, step;
    double x, y;

    /* input section */
    printf("Please enter the lower bound: ");
    scanf("%lf", &lo);
    printf("Please enter the upper bound: ");
    scanf("%lf", &hi);
    printf("Please enter the step size: ");
    scanf("%lf", &step);

    ...

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

46

Math Library Functions

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

```

...

/* computation and output section */
for(x = lo; x <= hi; x += step)
{
    y = f(x);
    printf("f(%10g) = %10g\n", x, y);
} /* rof */

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

/* EOF */

```

Math Library Functions

- Example session: `Function.c`

```

% vi Function.c
% gcc Function.c -o Function -Wall -ansi -lm
% Function
Please enter the lower bound: -0.5
Please enter the upper bound: 1.0
Please enter the step size: .1
f(   -0.5) =  0.877583
f(   -0.4) =  0.921061
f(   -0.3) =  0.955336
f(   -0.2) =  0.980067
f(   -0.1) =  0.995004
f(-2.77556e-17) =  1
f(    0.1) =  0.995004
f(    0.2) =  0.980067
f(    0.3) =  0.955336
f(    0.4) =  0.921061
f(    0.5) =  0.877583
f(    0.6) =  0.825336
f(    0.7) =  0.764842
f(    0.8) =  0.696707
f(    0.9) =  0.62161
f(    1) =  0.540302
%

```


Standard Library Functions

- Standard C library
 - standard library supplied with every C compiler
 - predefined standard functions
 - e.g. `printf()`, `scanf()`, etc.
- C library header files
 - input/output function declarations `#include <stdio.h>`
 - standard function declarations `#include <stdlib.h>`
 - time function declarations `#include <time.h>`
 - etc.
- C library linker file
 - contains standard function definitions (pre-compiled)
 - library file `libc.a`
 - compiler *automatically links* against the standard library (no need to supply extra options)

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

49

Standard Library Functions

- Functions declared in `stdlib.h` (partial list)
 - `int abs(int x);`
 - `long int labs(long int x);`
 - return the absolute value of a (long) integer `x`
 - `int rand(void);`
 - return a random value in the range 0 – `RAND_MAX`
 - `RAND_MAX` is a constant integer (e.g. 32767)
 - `void srand(unsigned int seed);`
 - initialize the random number generator with value `seed`
 - `void exit(int result);`
 - exit the program with return value `result`
 - `void abort(void);`
 - abort the program (with an error result)

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

50

Standard Library Functions

- Random number generation
 - Standard library provides *pseudo* random number generator
 - `int rand(void);`
 - Pseudo random numbers are a sequence of values seemingly random in the range 0 – `RAND_MAX`
 - Computer is a *deterministic* machine
 - Sequence will always be the same
 - Start of sequence is determined by *seed* value
 - `void srand(unsigned int seed);`
 - Trick: Initialize random sequence with current time
 - header file `time.h` declares function `unsigned int time()`
 - `time(0)` returns number of seconds since Jan 1, 1970
 - at beginning of program, use:
`srand(time(0));`

Standard Library Functions

- Program example: `Dice.c` (part 1/4)

```

/* Dice.c: roll the dice                                     */
/* author: Rainer Doemer                                    */
/* modifications:                                          */
/* 10/28/04 RD initial version                             */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

/* function definition */

int roll(void)
{
    int r;

    r = rand() % 6 + 1;
    /* printf("Rolled a %d.\n", r); */
    return r;
} /* end of roll */
...

```

Standard Library Functions

- Program example: `Dice.c` (part 2/4)

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    int i, n;
    int count1 = 0, count2 = 0, count3 = 0,
        count4 = 0, count5 = 0, count6 = 0;

    /* random number generator initialization */
    srand(time(0));

    /* input section */
    printf("Roll the dice: How many times? ");
    scanf("%d", &n);

    ...

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

53

Standard Library Functions

- Program example: `Dice.c` (part 3/4)

```

... /* computation section */
for(i = 0; i < n; i++)
{ switch(roll())
  { case 1:
    { count1++; break; }
    case 2:
    { count2++; break; }
    case 3:
    { count3++; break; }
    case 4:
    { count4++; break; }
    case 5:
    { count5++; break; }
    case 6:
    { count6++; break; }
    default:
    { printf("INVALID ROLL!");
      exit(10); }
  } /* hctiws */
} /* rof */
...

```

EECS10: Computational Methods in ECE, Lecture 6

(c) 2013 R. Doemer

54

Standard Library Functions

- Program example: `Dice.c` (part 4/4)

```

...

/* output section */
printf("Rolled a 1 %5d times.\n", count1);
printf("Rolled a 2 %5d times.\n", count2);
printf("Rolled a 3 %5d times.\n", count3);
printf("Rolled a 4 %5d times.\n", count4);
printf("Rolled a 5 %5d times.\n", count5);
printf("Rolled a 6 %5d times.\n", count6);

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

/* EOF */

```

Standard Library Functions

- Example session: `Dice.c`

```

% vi Dice.c
% gcc Dice.c -o Dice -Wall -ansi
% Dice
Roll the dice: How many times? 6000
Rolled a 1   963 times.
Rolled a 2   995 times.
Rolled a 3  1038 times.
Rolled a 4  1024 times.
Rolled a 5   984 times.
Rolled a 6   996 times.
% Dice
Roll the dice: How many times? 6000
Rolled a 1   977 times.
Rolled a 2  1043 times.
Rolled a 3  1012 times.
Rolled a 4  1001 times.
Rolled a 5   963 times.
Rolled a 6  1004 times.
%

```