EECS 10: Computational Methods in Electrical and Computer Engineering
Lecture 8

Brian Demsky
bdemsky@uci.edu

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

• Read Chapter 4 in the book
• Office hours are rescheduled for today:
  • 2-3 pm
  • If you can’t make these and need meet, email me to schedule a different time
Lecture 8: Overview

- Debugging
- Libraries
Debugging

- Problem: You write your program and it doesn’t work
- Debugging: Find the mistakes and fix them
- Finding bugs:
  - Test your program
- Two basic approaches:
  - Add printf statements to figure out what is going on
  - Tool-based debugging (i.e. gdb)
Debugging with Printf

- Two basic uses for printf statements:
  - Use printf statements to figure out where program is crashing
  - Use printf statements to figure out what values variables have
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`
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)

  - `double sqrt(double x);`  
    $\sqrt{x}$
  - `double pow(double x, double y);`  
    $x^y$
  - `double exp(double x);`  
    $e^x$
  - `double log(double x);`  
    $\log(x)$
  - `double log10(double x);`  
    $\log_{10}(x)$
  - `double ceil(double x);`  
    $\lceil x \rceil$
  - `double floor(double x);`  
    $\lfloor x \rfloor$
  - `double fabs(double x);`  
    $|x|$
  - `double fmod(double x, double y);`  
    $x \mod y$
Math Library Functions

- Functions declared in `math.h` (part 2/2)
  - `double cos(double x); cos(x)`
  - `double sin(double x); sin(x)`
  - `double tan(double x); tan(x)`
  - `double acos(double x); acos(x)`
  - `double asin(double x); asin(x)`
  - `double atan(double x); atan(x)`
  - `double cosh(double x); cosh(x)`
  - `double sinh(double x); sinh(x)`
  - `double tanh(double x); tanh(x)`
Math Library Functions

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

```c
/* 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 */

...
Math Library Functions

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

```c
... /* 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);

    ...
```
Math Library Functions

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

```c
...
/* 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`

```bash
% 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)
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)
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);`
Seeding the Random Number Generator

- Sometimes a repeatable sequence is good
  - Enables us to repeat numerical simulations
  - Useful for debugging
- Sometimes it is bad
  - Slot machines & other games
  - Encryption algorithms
- 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)

```c
/* 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)

```c
... /* 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);

    ...
Standard Library Functions

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

```c
... /* 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 */
...```

Standard Library Functions

• Program example: Dice.c (part 4/4)

```c
... /* 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**

```bash
% 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.
% 
```