

EECS 22: Advanced C Programming

Lecture 5

Presented by Che-Wei Chang

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

Lecture 5: Overview

- Review of the C Programming Language
 - Functions
 - Introduction and concept
 - Declaration, definition, and function call
- Hierarchy of Functions
 - Example program `Cylinder.c`
 - Function call graph
 - Function call trace
 - Function call stack
 - Long Jump
- Recursion
 - Concept of recursion
 - Example program `Fibonacci.c`

Review of the C Programming Language

- Functions
 - Support for essential 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 (often written by somebody else)

EECS22: Advanced C Programming, Lecture 5

(c) 2014 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

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

4

Functions

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

```
double CircleArea(double r);
```

 - function is named `CircleArea`
 - function takes one parameter `r` of type `double`
 - function returns a value of type `double`

EECS22: Advanced C Programming, Lecture 5

(c) 2014 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 CircleArea(double r)
{
    const double pi = 3.1415927;
    double a;
    a = pi * r * r;
    return a;
}
```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 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 = 10.0;
a = CircleArea(b);
```

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

EECS22: Advanced C Programming, Lecture 5

(c) 2014 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.

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

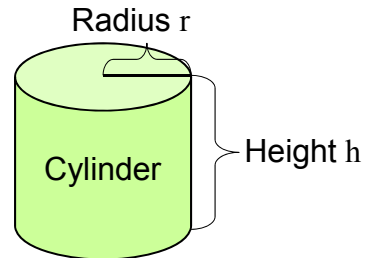
8

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

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

9

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);
}
...

```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

10

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);
}
...

```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

11

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 */

```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

12

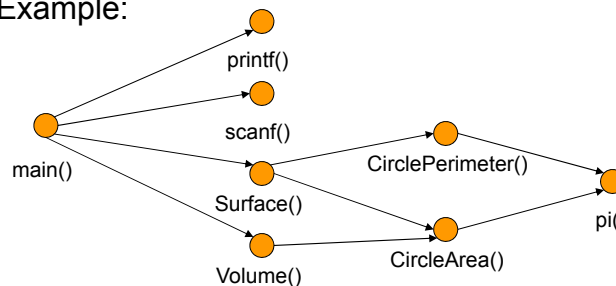
Functions

- 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.
%
```

Function Call Graph

- Graphical Representation of Function Calls
 - Directed Graph
 - Nodes: Functions
 - Edges: Function calls
 - Shows dependencies among functions
 - Example:



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()`

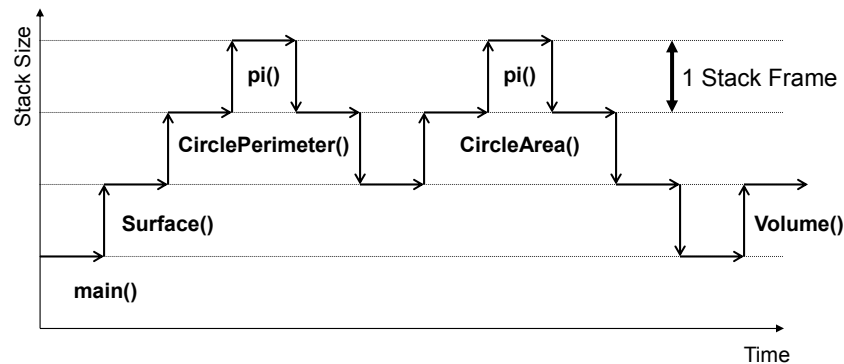
EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

15

Function Call Stack

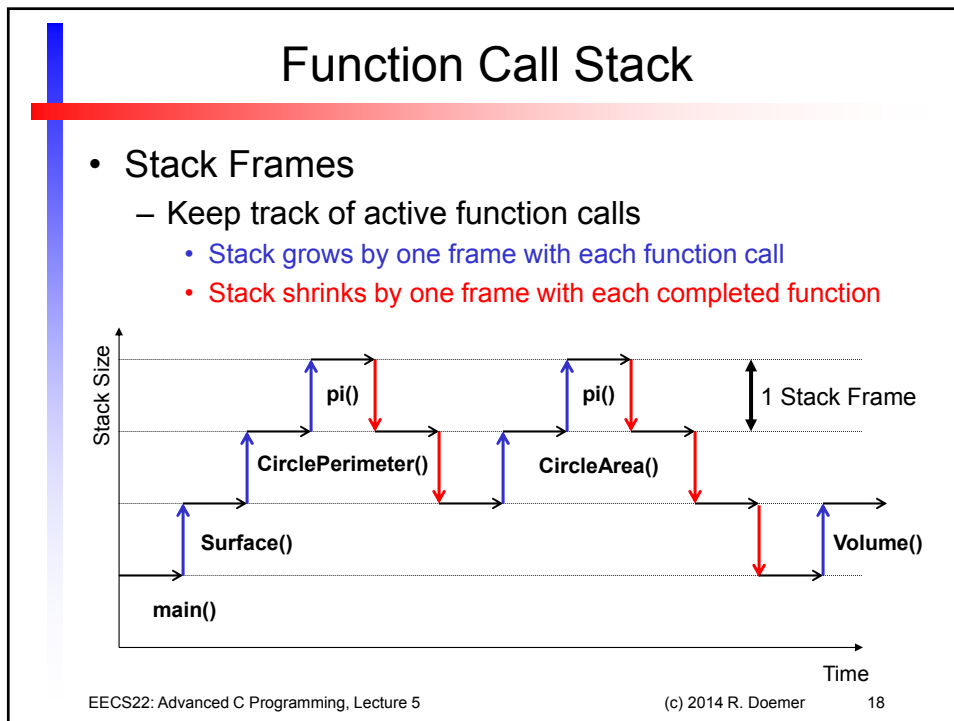
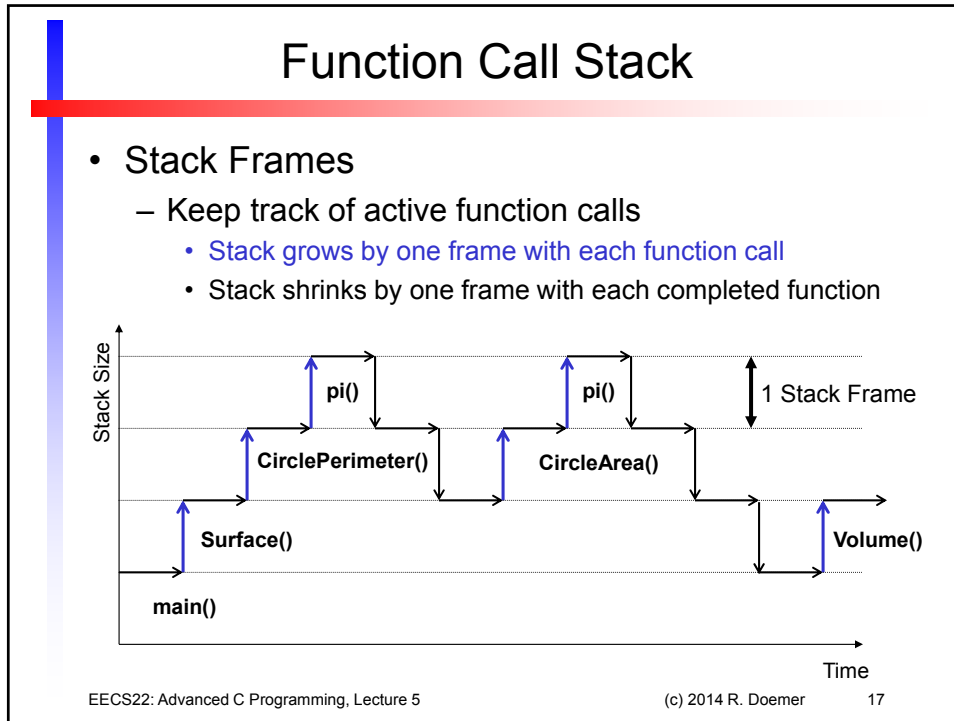
- Stack Frames
 - Keep track of active function calls
 - Stack grows by one frame with each function call
 - Stack shrinks by one frame with each completed function



EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

16



Non-Local Goto: Long Jump

- *Long Jump*: Returning to a previous stack frame
 - Useful, for example, when dealing with errors (or interrupts) in a low-level function of a program.
 - However, long jumps are hard to understand and maintain!
 - Same as goto, avoid long jumps, if possible!
 - `#include <setjmp.h>`

```
int setjmp(jmp_buf env);
```

 - saves current stack context in `env` for later use by `longjmp()`
 - stack context in `env` is valid until the function which called `setjmp()` returns
 - `void longjmp(jmp_buf env, int val);`
 - non-local jump (return) to a saved stack context `env`
 - `longjmp()` restores the stack context saved by `setjmp()`
 - after `longjmp()`, program execution continues as if the call of `setjmp()` had just returned the value `val`

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

19

Non-Local Goto: Long Jump

- *Long Jump*: Returning to a previous stack frame
- Example:

```
#include <setjmp.h>
jmp_buf env;          /* storage for stack context */
void error(void)      /* error, return to main! */
{
    longjmp(env, 1);
}
int main(void)
{
    if (setjmp(env)) /* store current stack context */
    { /* long jump arrives here! */
        return 10;
    }
    work(...); /* call tree can call error at any time */
    return 0;
}
```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

20

Recursion

- Introduction
 - *Recursion* is often an alternative to *Iteration*
 - Recursion is a very simple concept, yet very powerful
 - Recursion is present in nature
 - Trees have branches, which have branches, which have branches, ... which have leaves.
 - Recursion is traversal of hierarchy
 - *Traverse* (climb) a tree to the top:
 - start at the root
 - at a leaf, stop
 - at a branch, *traverse* one branch
 - *Traverse* a file system on a computer
 - start at the current directory
 - at a file, process the file
 - at a directory, *traverse* the directory

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

21

Recursion

- Recursive Function
 - Function that calls itself ...
 - ... directly, or
 - ... indirectly
- Concept of Recursion
 - Trivial *base case*
 - Return value defined for simple case
 - Example: `if (arg == 0) {return 1; }`
 - *Recursion step*
 - Reduce the problem towards the base case
 - Make a recursive function call
 - Example: `if (arg > 0) { return ...fct(arg-1); }`
- Termination of Recursion
 - Converging of recursive calls to the base case
 - Recursive call must be “simpler” than current call

```
int f(...)
{ ...
  f(...);
  ...
}
```

```
int a(...)
{ ...
  b(...);
  ...
}
int b(...)
{ ...
  a(...);
  ...
}
```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

22

Recursion

- Example: Fibonacci series
 - Sequence of integers
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...
 - Mathematical properties
 - The first two numbers are 0 and 1
 - Every subsequent Fibonacci number is the sum of the previous two Fibonacci numbers
 - Ratio of successive Fibonacci numbers is ...
 - ... converging to constant value 1.618...
 - ... called *Golden Ratio* or *Golden Mean*
 - Recursive definition:
 - Base case: $fibonacci(0) = 0$
 $fibonacci(1) = 1$
 - Recursion step: $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

23

Recursion

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

```

/* Fibonacci.c: example demonstrating recursion */
/* author: Rainer Doemer */
/* modifications: */
/* 11/14/04 RD initial version */

#include <stdio.h>

/* function definition */
long fibonacci(long n)
{
    if (n <= 1) /* base case */
    { return n;
      } /* fi */
    else /* recursion step */
    { return fibonacci(n-1) + fibonacci(n-2);
      } /* esle */
} /* end of fibonacci */

/* main function */
...

```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

24

Recursion

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

```

...
int main(void)
{
    /* variable definitions */
    long int n, f;

    /* input section */
    printf("Please enter value n: ");
    scanf("%ld", &n);

    /* computation section */
    f = fibonacci(n);

    /* output section */
    printf("The %ld-th Fibonacci number is %ld.\n", n, f);

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

/* EOF */

```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

25

Recursion

- Example session: `Fibonacci.c`

```

% cp Factorial.c Fibonacci.c
% vi Fibonacci.c
% gcc Fibonacci.c -o Fibonacci -Wall -ansi
% Fibonacci
Please enter value n: 1
The 1-th Fibonacci number is 1.
% Fibonacci
Please enter value n: 10
The 10-th Fibonacci number is 55.
% Fibonacci
Please enter value n: 20
The 20-th Fibonacci number is 6765.
% Fibonacci
Please enter value n: 30
The 30-th Fibonacci number is 832040.
% Fibonacci
Please enter value n: 40
The 40-th Fibonacci number is 102334155.
%

```

EECS22: Advanced C Programming, Lecture 5

(c) 2014 R. Doemer

26