

EECS 22: Advanced C Programming

Lecture 6

Rainer Dömer

doemer@uci.edu

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

Lecture 6: Overview

- Functions
 - Terms and concepts
 - Scope
- Variable Lifetimes
- Memory Organization
 - Memory segmentation
 - Memory errors
- Storage Classes
- Program Example **Fibonacci2**

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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

3

Scope

- *Scope of an identifier*
 - Portion of the program where the identifier can be referenced
 - aka. *accessability, visibility*
- Variable scope examples
 - Global variables: *file scope*
 - Declaration outside any function (at global level)
 - Scope in entire translation unit 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)

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

4

Scope 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;
}
```

Header file inclusion

Function declarations

Global variables

Function definition
Local variableFunction definition
Local variableFunction definition
Local variable

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

5

Scope 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.

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

6

Scope 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()

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

7

Scope 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()

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

8

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

9

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

10

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

11

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

12

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

13

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

14

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

15

Scope 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

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

16

Variable Lifetimes

- Lifetime of Variables
 - Begins with *allocation*
 - Assignment of a (new) address in memory
 - Ends with *deallocation*
 - Memory is freed, address is marked as unused
 - Initialization: first access must be a write-access
 - Otherwise, variable value is undefined!
 - Access to a variable before or after its lifetime results in undefined behavior!
 - Don't confuse *Variable Lifetime* with *Variable Scope*!
 - Variable Scope is determined at compile time
 - Variable Lifetime is determined at run time

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

17

Variable Lifetimes

- Lifetime of Variables: Examples
 - Global variables (storage class **static**, **extern**)
 - From program start to end
 - Local variables (storage class **register**, **auto**)
 - From beginning of execution of the compound statement
 - Stack frame entry
 - To leaving the compound statement
 - Stack frame exit
 - Function parameters (storage class **register**, **auto**)
 - From beginning of function call
 - To returning from the function call
 - Dynamically allocated objects (via **malloc()**, **free()**)
 - From successful return of **malloc()**
 - To call of **free()**

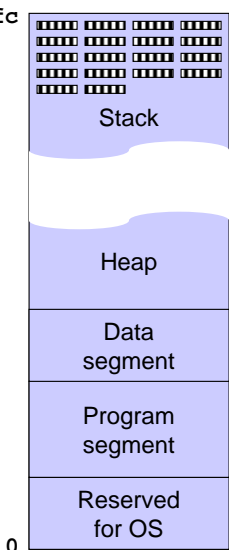
EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

18

Memory Organization

- Memory Segmentation
 - typical (virtual) memory layout on processor with 4-byte words and 4 GB of memory
 - Stack
 - grows and shrinks dynamically (from top)
 - contains function call hierarchy
 - stores stack frames with local variables
 - Heap
 - “free” storage
 - dynamic allocation by the program
 - Data segment
 - global (`static`) variables
 - Program segment
 - stores program text (binary code)
 - Reserved area for operating system



ffff fffc

Stack

Heap

Data segment

Program segment

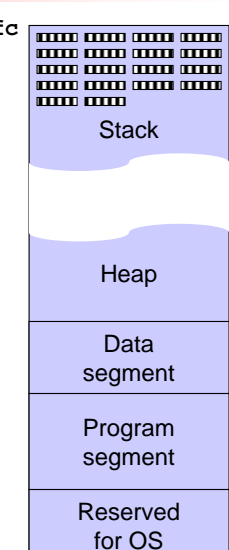
Reserved for OS

0

EECS22: Advanced C Programming, Lecture 6 (c) 2012 R. Doemer 19

Memory Organization

- Memory Segmentation
 - typical (virtual) memory layout on processor with 4-byte words and 4 GB of memory
- Memory errors
 - *Out of memory*
 - Stack and heap collide
 - *Segmentation fault*
 - access outside allocated segments
 - e.g. access to segment reserved for OS
 - *Bus error*
 - mis-aligned word access
 - e.g. word access to an address that is not divisible by 4



ffff fffc

Stack

Heap

Data segment

Program segment

Reserved for OS

0

EECS22: Advanced C Programming, Lecture 6 (c) 2012 R. Doemer 20

Storage Classes

- C Language distinguishes 2 Storage Classes
 - (but uses 5 keywords and a default, depending on scope)
 - Automatic (i.e. on the stack)
 - **auto** local variable, on stack (default)
 - **register** local variable, in register (preferred) or on stack
 - Static (i.e. in the data segment)
 - **static** static variable in data segment
 - **extern** declaration of global variable in data segment
- At compile-time, a 3rd “storage class” exists
 - **typedef** definition of an alias for a type at compile time (no storage)

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

21

Storage Classes

Keyword	Global Scope	Local Scope
(none)	Global variable/function in data segment (ext. linkage)	Local variable on stack
auto	n/a	Local variable on stack
register	n/a	Local variable on stack or in register (preferred)
static	Global variable/function in data segment (int. linkage)	Local variable in data segment
extern	Decl. of global variable/function in data segment (ext. linkage)	Decl. of global variable/function in data segment (ext. linkage)
typedef	Alias for a type at compile time (no storage in memory)	Alias for a type at compile time (no storage in memory)

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

22

Program Example

- Example Revisited: Fibonacci series
 - Recursive definition:
 - Base case: $fibonacci(0) = 0, fibonacci(1) = 1$
 - Recursion step: $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$
 - Mathematical properties:
 - The first two numbers are 0 and 1
 - Every subsequent number is the sum of the *previous* two
 - Problem:
 - Recursive calculation time grows exponentially!
 - Idea:
 - If we remember previously calculated numbers, we can calculate the next number immediately!
 - Whenever a new number is calculated, keep it stored in a static array in memory
 - When a number is present in the memory, just look it up

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

23

Program Example

- Program example: `Fibonacci2.c` (part 1/3)

```

/* Fibonacci2.c: example demonstrating recursion */
/* author: Rainer Doemer */
/* modifications: */
/* 11/09/11 RD version with 'static' memory */
/* 11/14/04 RD initial version */

#include <stdio.h>

#define MEM_SIZE 100

/* function definition */

...

```

EECS22: Advanced C Programming, Lecture 6

(c) 2012 R. Doemer

24

Program Example

- Program example: `Fibonacci2.c` (part 2/3)

```

long fibonacci(long n)
{
    static long fib[MEM_SIZE] = {0,1}; /* memory */
    if (n <= 1) /* base case */
    {
        return n;
    } /* fi */
    else /* previously calculated results */
    {
        if (n < MEM_SIZE && fib[n])
        {
            return fib[n];
        } /* fi */
        else /* recursion step */
        {
            long f;
            f = fibonacci(n-1) + fibonacci(n-2);
            if (n < MEM_SIZE)
            {
                fib[n] = f; /* remember this */
            } /* fi */
            return f;
        } /* esle */
    } /* esle */
} /* end of fibonacci */
...

```

EECS

5

Program Example

- Program example: `Fibonacci2.c` (part 3/3)

```

...
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 6

(c) 2012 R. Doemer

26

Program Example

- Example session: **Fibonacci2.c**

```
% cp Fibonacci.c Fibonacci2.c
% vi Fibonacci2.c
% gcc Fibonacci2.c -o Fibonacci2 -Wall -ansi
% Fibonacci2
Please enter value n: 20
The 20-th Fibonacci number is 6765.
% Fibonacci2
Please enter value n: 30
The 30-th Fibonacci number is 832040.
% Fibonacci2
Please enter value n: 40
The 40-th Fibonacci number is 102334155.
% Fibonacci2
Please enter value n: 50
The 50-th Fibonacci number is 12586269025.
% Fibonacci2
Please enter value n: 60
The 60-th Fibonacci number is 1548008755920.
%
```