UNIVERSITY OF MASSACHUSETTS
DARTMOUTH

# ECE160: Foundations of Computer Engineering I

## Lecture #16 – Functions (III)

Instructor: Dr. Liudong Xing
SENG-213C, lxing@umassd.edu
ECE Dept

# Administrative Issues

- Lab#7 starts today
  - Due Wednesday, March 15, 5pm

- Today's topics
  - Finish L#15 (Function -- Exercise 5a and 5b)
  - Then L#16 (Parameter passing & Standard library functions)

# Review of Lectures #15

- The general formats for functions
  - defining
  - declaring
  - calling

- Steps for writing a programmer-defined functions
  - Think what the prototype will look like. Place the prototype before main().
  - Think where you will call the function in main() and what you will pass to it.
  - Think what the function will do.

# Outline

- Parameters passing
  - Pass by value
  - Pass by reference
- C standard library functions

# Pass by Value

- A copy of the data (argument's value) is passed to the called function.
- The function can not modify the original variable's value in the caller.
- This is what we have done so far.

# Example (Pass by Value)

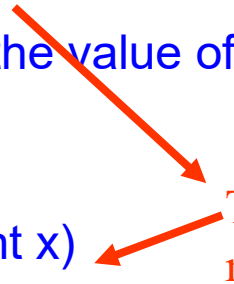
What is the output of the program?

```
#include <stdio.h>
void test(int x);

void main(void)
{
    int a;
    a =2;
    test(a);
    printf("the value of a after call is %d\n", a);
}

void test(int x)
{
    x = x + 5;
}
```

the value of a after call is 2



The value of a is copied into the memory cell reserved for x in the region of memory for test function

# Pass by Reference

- The called function can modify the original variable's value in the caller.
- Any reference to a parameter is the same as a reference to the variable in the calling function
- It uses the address operator (&) and indirection operator (*).

# Example (Pass by Reference)

```
#include <stdio.h>
void test(int *x);

void main(void)
{
    int a;
    a =2;
    test(&a);
    printf(" the value of a after call is %d\n", a);
}

void test(int *x)
{
    *x = *x + 5;
}
```

In a function prototype or header, * means the variable following * is to hold an address

& means the address of , a copy of the address of variable a is put into memory cell reserved for x in the memory region reserved for the variables of test function

**the value of a after call is 7**

In a C statement, * is an unary operator (when it is not meant to mean multiplication), meaning to go to the address stored in the variable following * and get the value at that address or put a value at that address

→ add 5 to the contents of memory cell assigned to variable a

# Example
# How to swap two numbers

```
#include <stdio.h>

void swapp(int n1, int n2);

void main(void)
{
    int x=2;
    int y=3;

    swapp(x, y);

    printf("x and y are:%d %d\n",
     x, y);
}
```

```
void swapp(int n1, int n2)
{
    int temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
    return;
}
```

***Wrong Program!***

# Example
# How to swap two numbers (Right Program)

```
#include <stdio.h>

void swapp(int *n1, int *n2);

void main(void)
{
    int x=2;
    int y=3;

    swapp(&x, &y);

    printf("x and y are:%d %d\n",
     x, y);
}
```

```
void swapp(int *n1, int *n2)
{
    int temp;
    temp = *n1;
    *n1 = *n2;
    *n2 = temp;
    return;
}
```

# Outline

- Parameters passing
  - Pass by value
  - Pass by reference
- **C standard library functions**
  - Mathematical functions
  - Random number generation functions: srand(), rand() Lecture#17
  - Character functions Lecture#17

# Standard Library Functions

- C has a rich collection of functions whose definitions have been written and are ready to be used in your programs

# Note!

- In general, you write a programmer-defined function *iff* no such function is in the C library.

- The reason is that the C library functions were written by professional programmers and have been used and tested many times. Therefore, they are more reliable, more efficient, and more portable than the functions you can write!

# Using Standard Library Functions

- To use them, include their prototype declarations in the program

- Their prototypes are grouped into header files
  - Input/output functions (printf, scanf) → stdio.h
  - Mathematical functions → math.h, stdlib.h
  - General utility functions → stdlib.h
  - Etc…

- Use **include** statement to include the header files
  - Example: **#include <stdio.h>**

# C Standard Mathematical Functions

most in math.h

abs() and labs() are in stdlib.h

# abs/labs/fabs

- int  abs (int number);
  - returns the absolute value of an integer
  - in **stdlib.h**
- long labs(long number);
  - returns the absolute value of a long integer
  - in **stdlib.h**
- double fabs(double number);
  - returns the absolute value of a double
  - in **math.h**

# ceil/floor

- double ceil (double number);
  - returns the  smallest integral value greater than or equal to a number.
  - Example:

    ceil(2.01)

    ceil (-2.3)

- double floor (double number);
  - returns the largest integral value that is equal or less than a number.
  - Example:

    floor(-1.1)

    floor(1.9)

# pow/sqrt

- double pow (double x, double y);
  - return the value of x raised to the power y, i.e., $x^y$
  - Example: pow(3.0, 4.0) → 81.0

- double sqrt(double number);
  - returns the square root of a number.
  - Example: sqrt(25.0) → 5.0

# Exercise

- What is the value of the following expressions:

```
x=  ceil (-fabs(-10.2) + floor(-3.3) );

y =  fabs(ceil(3.1) + floor(-100.3) + 12);
```

# Summary of Lectures #16

- Two ways to pass parameters to functions
  - Passing by value: a copy of the data (argument's value) is passed to the called function.
  - Passing by reference: any reference to a parameter is the same as a reference to the variable in the calling function
- C has a rich collection of standard library functions which are ready to be used in your programs
  - Mathematical functions

**To continue in Lecture#17**

  - Random number generation functions: srand(), rand()
  - Character functions

# Things To Do

- Review lecture notes
- Run the programs on Slides 6, 8, 9, 10 to verify the results

# Next Topics

- C standard library functions (Cont'd)
- Recursion