UNIVERSITY OF MASSACHUSETTS
DARTMOUTH

# ECE160: Foundations of Computer Engineering I

## Lecture #17 -- Functions (IV): C Standard Library Functions & Recursions

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

# Administrative Issues

- Lab#7 starting on Monday, March 13
  - Due **5pm, Wednesday, March 15**
- Homework#4 assigned today
  - Due **9am, Wednesday, March 22**
- Today's topics
  - C standard library functions (Cont'd)
  - Repetitive algorithms

# Review 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
  - More in Lecture#17

# Review Questions (True/False)

- _____ The value of floor(-3.7) is –3
- _____ The value of abs(7) is -7
- _____ The value of expression ceil(1.234*100+0.3)/100 is 1

# Outline

- **C standard library functions (Cont'd)**
  - Random number generation functions: srand(), rand()
  - Character functions
- Repetitive processes
  - Iterations
  - Recursions

# General Library Functions
# (most in stdlib.h)

## Random Number Generation Functions

srand()

rand()

# Seed Random Generation Function *srand()*

- Prototype:

```
void srand (unsigned int seed);
```

- Generates the first seed for a pseudorandom number series.
  - a pseudorandom number series is a repeatable series of numbers with random properties.
  - a seed is a variable used by **rand()** to calculate the next number in the series
  - a large prime number is preferred

```
srand(997);
```

# *srand()* (cont'd)

- To generate a truly random number series, the seed must be a random number!
  - Use a seed that is a function of current date or time of day

```
srand(time(NULL));
```

The C library function **time() in time.h** can be used, which returns the time since the Epoch (00:00:00 UTC, January 1, 1970), measured in seconds.

  - A different series can be got each time you run the program

Note: *srand* is called only once for each random number series, usually only once in a program!

# Random Number Generator *rand()*

- Prototype:

  int rand (void)

- Returns a pseudorandom integer between 0 and RAND_MAX.
- RAND_MAX is defined in the standard library as the largest number that *rand()* can generate (>=32767)
- Each call generates the next number in a random number series
- Use seed 1 if *srand()* is not called before the 1st call to *rand()*

# Exercise (1)

- Write a program that generates 3 random numbers and prints them out.
  - Remember to include the “stdlib.h” and “time.h” files.

# Solution

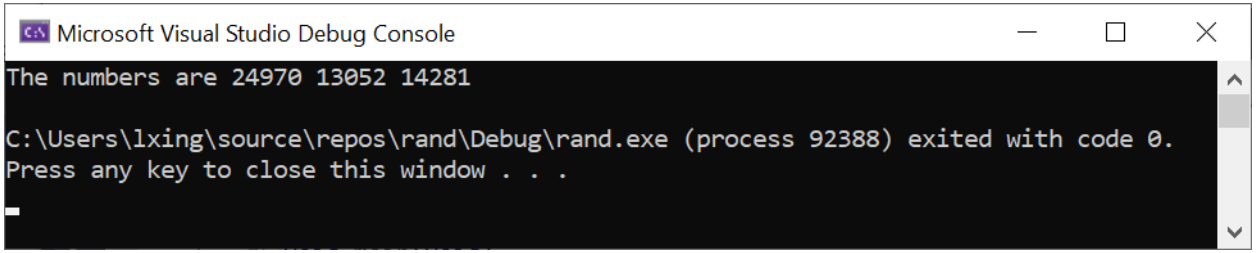
```
#include "stdio.h"
#include "stdlib.h"
#include "time.h"

void main(void)
{
    int rand1;
    int rand2;
    int rand3;

    srand(time(NULL));

    rand1 = rand();
    rand2 = rand();
    rand3 = rand();
    printf("The numbers are %d %d %d\n", rand1, rand2, rand3);
}
```

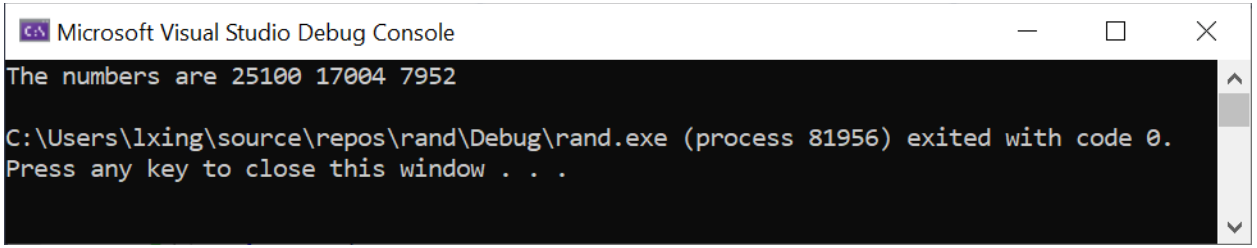
***Exercises:***

Try it with and without calling the *srand()* function;

Run the program twice for each case and compare the results

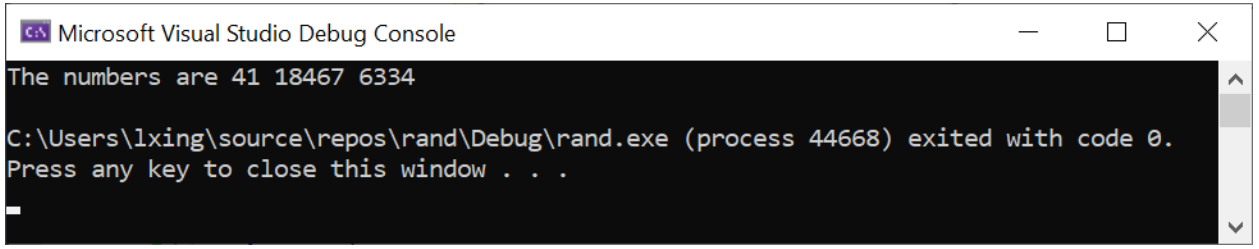
- With srand(time(NULL));

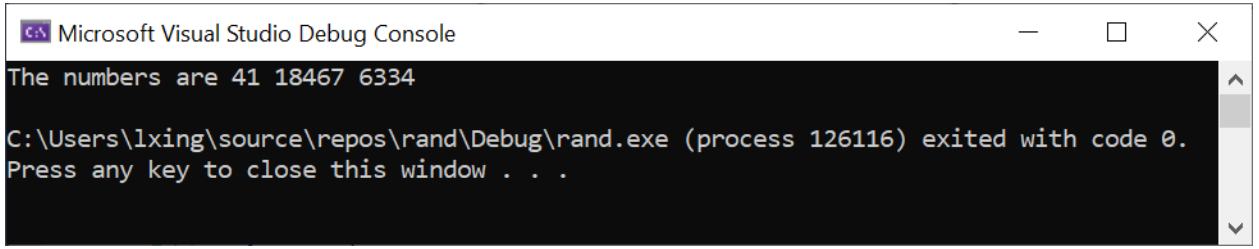



***Every time you run the program you get different three random numbers***

- Without srand(time(NULL));




***You always get the same three random numbers***

# Scaling Random Numbers

- To generate random numbers in a narrower range than provided by library

- Scaling is done using the modulus operator.

  rand() % M

  returns random numbers in the range 0 to M-1.

- Example:

  rand() % 31

  → random numbers in range 0~30

# Scaling Random Numbers (Cont'd)

- To scale numbers in the range min ~ max, we scale like this:

  rand() %((max + 1)-min) + min

- Example:

  rand() % ((30+1)-20)+20

  → rand() % 11 + 20

  random numbers in range 20~30

# Exercise (2)

What is the range of the following random numbers?

```
rand() % 11
rand()%10 +10
rand()%5-1
```

# Modify Exercise (3)

- Modify the program in the random number generation example (Slide 11) so that the program generates random numbers in the range 100-200.

# Standard Characters Functions (in ctype.h)

- Classifying functions
- Converting functions

# Classifying Functions

- Examine a character and tell its type
- Format: int is…(int testchar);
- Return either 1 (true) or 0 (false)
- Examples:
  - int isalpha(int c); tests whether c belongs to the alphabetical set (A…Z, a…z)
  - int islower(int c); tests whether it is a lower case character
  - int isupper(int c); tests whether it is an upper case character.
  - int isdigit(int c); tests whether it is a digit (0…9).

# Character Conversion Functions

- Convert a character from one type to another
- Format: int to….(int oldchar);
- Return an integer that is the value of the converted character
- Examples:
  - int toupper(int c); converts the input character to an upper case character.
  - int tolower(int c); converts the input character to a lower case character.

# Exercise (4)

```
#include "stdio.h"
#include "ctype.h"
void main(void)
{
      char c;
      int m;
      printf("Please enter a character\n");
      scanf_s("%c", &c);
      if(isdigit(c))   printf("You entered a digit\n");
      if(isalpha(c))
          {
               printf("You entered a letter\n");
               if(isupper(c))  printf("You entered an uppercase letter\n");
               if(islower(c))
               {      printf("You entered a lowercase letter\n");
                      m = toupper(c);
                      printf("I converted the character to uppercase %c\n",m);
               }
          }
}
```

***Testing exercises:***

Run the program with the following inputs and under the results:

A

9

f

# Review Questions (True/False)

- _____ The character classifications are found in the standard library header file stdlib.h
- _____ To check if a character is uppercase, the toupper function is used
- _____The expression rand()%20-6 can create a random number in the range –6 ~ 14

# Outline

- ✓ C standard library functions (Cont'd)
  - ✓ Random number generation functions: srand(), rand()
  - ✓ Character functions
- **Repetitive processes**
  - Iterations
  - Recursions

# Repetitive Algorithms

- Two approaches to writing repetitive algorithms
  - Using loops (for, while, do…while)
    - Iterative way
    - A repetitive function is defined iteratively whenever the definition involves only the parameter(s) and not the function itself
  - Using recursion: a repetitive process where a function calls itself.

# Example

- Write a function to compute a factorial:

$$factorial(n) = \begin{cases} 1 & \text{if} \quad n = 0 \\ n*(n-1)*...*2*1 & \text{if} \quad n > 0 \end{cases}$$

  - product of the integral values from 1 to n
  - Example:

factorial(3) = 3*2*1=6

# Implementation #1 (Iterative)

```
#include "stdio.h"
long factorial(int n);
void main(void)
{
      int a;
      long f;
      printf("Enter a number \n");
      scanf_s("%d",&a);

      f =factorial(a);

      printf("The factorial is %d \n", f);

}
```

?

Define factorial () using a *for* loop

$$factorial(n) = \begin{cases} 1 & \text{if} \quad n = 0 \\ 1*2*...*(n-1)*n & \text{if} \quad n > 0 \end{cases}$$

# Repetitive Algorithms (Revisit)

- Two approaches to writing repetitive algorithms
  - Using loops (for, while, do…while)
    - Iterative way
    - A repetitive function is defined iteratively whenever the definition involves only the parameter(s) and not the function itself
  - **Using recursion: a repetitive process where a function calls itself.**

# Recursive Definition

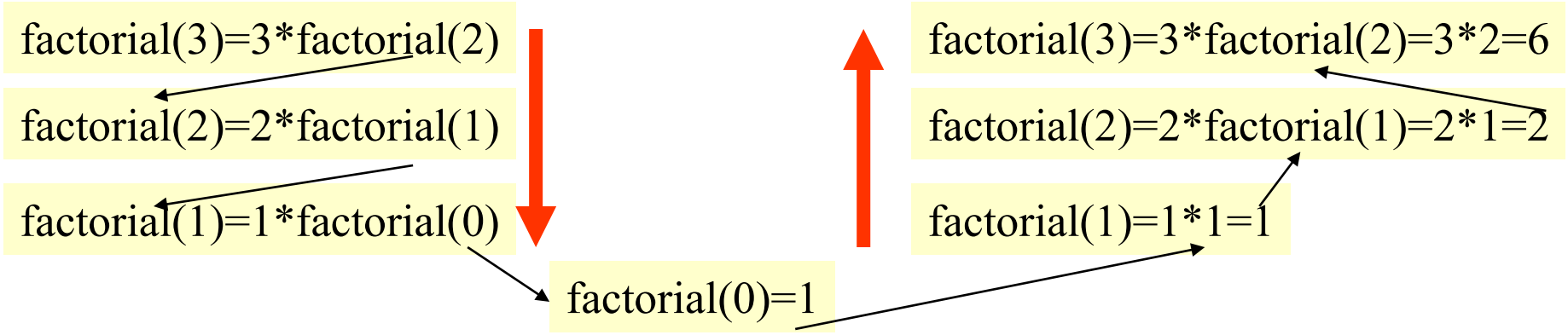
- Recursive definition
  - A repetitive function is defined recursively whenever the function appears within the definition itself.

- Example: the computation of a factorial:

$$factorial(n) = \begin{cases} 1 & \text{if} \quad n = 0 \\ n * factorial(n-1) & \text{if} \quad n > 0 \end{cases}$$

# Example: Decomposition of factorial(3)

$$factorial(n) = \begin{cases} 1 & \text{if} \quad n = 0 \\ n * factorial(n-1) & \text{if} \quad n > 0 \end{cases}$$



# Note!

- Recursive solution involves a two-way journey
  - First we decompose the problem from top to bottom
  - Then we solve it from bottom to top

- Base case:
  - The statement that "solves" the problem: factorial(0)
  - Every recursive function must have a base case
  - Once the base case has been reached, the solution begins

# Implementation #2 (recursive)

```
#include "stdio.h"
long factorial(int n);
void main(void)
{
      int a;
      long f;
      printf("Enter a number \n");
      scanf_s("%d",&a);

      f =factorial(a);

      printf("The factorial is %d \n", f);

}
```

?

Define factorial () using recursion

$$factorial(n) = \begin{cases} 1 & \text{if} \quad n = 0 \\ n * factorial(n-1) & \text{if} \quad n > 0 \end{cases}$$

# Note!

- Every recursive call must either solve part of the problem or reduce the size of the problem

- Rules for designing a recursive function:
  - First determine the base case
  - Then determine the general cases (other cases)
  - Combine the base case and general case into a function

# Exercise (5a)

- Write a **recursive** function that generates Fibonacci numbers
  - Named after Leonardo Fibonacci (an Italian mathematician)
  - A series in which each number is the sum of the previous two numbers
  - Example:

    0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,……

*Rules for designing a recursive function:*

First determine the base case

Then determine the general cases (other cases)

Combine the base case and general case into a function

# Exercise (5b)

- Write an **iterative** function that generates Fibonacci numbers using a **for loop**

# Summary of Lectures #17

- C standard library functions (2)
  - Random number generation functions: srand(), rand()
  - Character functions
- Two approaches to writing repetitive algorithms
  - Using loops (for, while, do…while; iterative way)
  - Using recursion: is a repetitive process where a function calls itself

# Things To Do

- Review lecture notes
- Run the programs on Slides 11, 16, 20, 25, 30 (Refer to the Solution file for complete programs if they are not available in the lecture)

# Next Topics

- Files