UNIVERSITY OF MASSACHUSETTS
DARTMOUTH

ECE160: Foundations of Computer Engineering I

Lecture #15 – Functions (II)

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

Administrative Issues (3/3, Friday)

- Lab#6 solution posted
- Homework#3 due **Today!**

- No classes next week. Have a great and safe Spring break!

Review of Lectures #14

- A function is an independent module that somebody calls it in order to perform a specific task.
- One reason for defining a function is to avoid writing the same group of C statements over and over again.
- Every C program contains one and only one main()
- Type of functions is the type of the data they return.
- Functions must be declared before being used in a program
- Information can be passed between a function and the function that calls it
- A list of common programming errors about functions

Outline

- Programmer-defined functions
  - Function definition
  - Function declaration
  - Function calling
  - Suggested steps

Function Definition

Format of A Function Definition

```
return_value_type   function_name(parameter_list)
{
    Local variables declaration and definition
    Statements
}
```

function header

function body

- return_value_type
  - the data type of the result returned to the caller: e.g., void, int, float, char
- function_name is any valid identifier
- parameter_list
  - a comma separated list specifying the parameters received by the function when it is called

Note!

- DO NOT use a semicolon at the end of the function header definition.
- The function body must be enclosed within a pair of braces!
- Ensure that what you are returning from the function matches the return type of the function.
- The type of each function argument must be ***individually*** defined in the parameter list.
- DO NOT define a function inside another function.

return statements

- Three ways to return program control from a called function to the point at which a function was invoked:
  - **For functions that do not return a result to the caller, control is returned**
    - **Way 1**: When the function-ending right brace } is reached
    - **Way 2**: By executing statement

      ```
      return;
      ```
  - **For functions that do return a result to the caller,**
    - **Way 3:** The statement returns the value of the expression to the caller

      ```
      return expression;
      ```

An Example of Function Definition

```
/* defining the Function */
float average_2(int num1, int num2)
 {
  float local_average;
  local_average =  (num1 + num2)/2.0;
  return local_average;
 }
```

Way #3

Ensure that what you are returning from the function matches the return type of the function.

Function Declaration

Function Declarations (I)

- Through the **function prototype** statements

  return_value_type function_name(parameter_list);

  – same as the function header, but with a semicolon at the end

  – Parameter names are not necessary, but we prefer to include them for documentation purpose

  – Example:

  float average_2(int num1, int num2);

  float average_2(int, int);

Function Declarations (II)

```
float average_2(int num1, int num2);
```

- A function prototype tells the compiler
  - The type of data returned by the function,
  - The number of parameters the function expects to receive,
  - The types of parameters,
  - The order in which the parameters are expected

- The compiler use them to validate function calls

Note!

- Make sure that the function prototype matches exactly the function's definition (return type, function name, number, types, and order of arguments). Otherwise, you will get a compilation error.

- Put a semicolon at the end of the function prototype.

Exercise (1)

- Find errors, if any, in the following function prototypes
  - int func1(int a, void);
  - float func2 (int a; int b)
  - void func3(int, double, int);
  - int func4(a,b,c);
  - void func5(int n, double, int m);
  - void func6(int,a, int,b, float,c);

Function Calls

Calling Functions

- Format

  function_name(parameter_expression_list)

  – Each expression in the list is commonly a single variable or constant

  – Separated by commas

  – Total number of expressions must equal to the number of arguments in the function prototype

- A function call *transfers program control* and *passes the values* from the caller to the function

Note!

- When the function has no arguments, remember to put the parentheses when you call them

- It is a logic error to pass the values to the parameters in the wrong order.

Exercise (2)

- What is the output from this program?

```
#include <stdio.h>
void NumOut(int m, float n); /*function declaration*/
void main (void)
{
        int a=1, b=2, c=3;
        float r=1.32, s=3.23, t=0.32;
        NumOut (a,b); /*function call*/
        NumOut (r,s);  /*function call*/
}
void NumOut (int m, float n) /*function definition*/
{
        printf("m=%d, n=%f\n", m, n);
}
```

Exercise (3)

- Find and correct errors, if any, in the following program and then specify the output from this program

```
#include <stdio.h>
/*function declaration*/
int func_sum(int, int)

void main (void);
{
        int a=3, b=7, c=6;
        float u=6.372, v=1.23;
        /*function calls*/
        printf("%d\n", func_sum1(a,b));
        printf("%d\n", func_sum2(a,u));
        printf("%d\n", func_sum3(b,c,u));
}

/*function definition*/
void func_sum(int m, int n);
{
        return m+n;
}
```

Agenda

- Programmer-defined functions
  - Function definition
  - Function declaration
  - Function calling
  - Suggested steps

What to do when you are asked to write a function

1. Think what the prototype will look like. Place the prototype before main().
2. Think where you will call the function in main() and what you will pass to it.
3. Think what the function will do.

Make sure that the function prototype matches exactly the function's definition (return type, function name, number, types, and order of arguments).

Exercise (4)

- Write a program that reads an integer number from the keyboard, multiplies it with 300 and prints out the result on the screen.

- Requirements:
  - Instead of doing a simple multiplication in main(), as

    a = 300 *a;

    do the multiplication of the variable that was read in, inside a function called amplifier.

```
#include "stdio.h"

void main(void)
{
    int a;
    printf("Enter a number\n");
    scanf("%d",&a);

    printf("Here is an amplified %d\n",a);

}
```

1. Think what the prototype will look like. Place the prototype before main()
   - –The type of data returned by the function: int
   - –The number of parameters the function expects to receive: 1
   - –The types of parameters: int
   - –The order in which the parameters are expected.

2. Think where you will call the function in main() and what you will pass to it.

3. Think what the function will do.

```
#include "stdio.h"
int amplifier(int num);

void main(void)
{
     int a;
     printf("Enter a number\n");
     scanf("%d",&a);

    a = amplifier(a);

     printf("Here is an amplified %d\n",a);

}
```

1. Think what the prototype will look like. Place the prototype before main()
   - –The type of data returned by the function: int
   - –The number of parameters the function expects to receive: 1
   - –The types of parameters: int
   - –The order in which the parameters are expected.

2. Think where you will call the function in main() and what you will pass to it.

```
int amplifier(int num)
{
          int k;
          k = num*300;
          return k;
}
```

3. Think what the function will do.

Or simply:

```
int amplifier(int num)
{
          return num*300;
}
```

Exercise (5a)

- Implement a calculator that
  1) reads a character (indicating the operation to be performed) from the keyboard: +, -, *
  2) reads two integer numbers (operands) from the keyboard
  3) operates on the input numbers
  4) outputs the result

Calculator without using programmer -defined functions:

1) reads a character (indicating the operation to be performed) from the keyboard: +, -, *
2) reads two integer numbers (operands) from the keyboard
3) operates on the input numbers
4) outputs the result

```
#include <stdio.h>
void main(void)
{
    char c;
    int a, b;
    int sum, difference, product;
    printf("Please enter the operation:\n");
    scanf("%c",&c);
    printf("Please enter two integers\n");
    scanf("%d%d",&a,&b);
    switch(c)
    {
        case '+': printf("This is an addition\n");
                  sum=a+b;
                  printf("The result is %d\n", sum);
                  break;
        case '-': printf("This is a subtraction\n");
                  difference=a-b;
                  printf("The result is %d\n", difference);
                  break;
        case '*': printf("This is a multiplication\n");
                  product=a*b;
                  printf("The result is %d\n", product);
                  break;
        default:  printf("Operation not defined\n");
     }
}
```

Exercise (5b)

- Implement a calculator that
  1) reads a character (indicating the operation to be performed) from the keyboard: +, -, *
  2) reads two integer numbers (operands) from the keyboard
  3) operates on the input numbers
  4) outputs the result

- Requirements:
  - uses a **switch** statement and **three user-defined functions** to implement the calculator.

```
#include <stdio.h>
void main(void)
{
        char c;
        int a, b;
        int sum, difference, product;
        printf("Please enter the operation:\n");
        scanf("%c",&c);
        printf("Please enter two integers\n");
        scanf("%d%d",&a,&b);
        switch(c)
        {
                case '+': printf("This is an addition\n");
                  add(a,b);     sum=a+b;
                                printf("The result is %d\n", sum);
                                break;
                case '-': printf("This is a subtraction\n");
                  subtract(a,b);  difference=a-b;
                  break;        printf("The result is %d\n", difference);
                                break;
                case '*': printf("This is a multiplication\n");
                                product=a*b;
                  multiply(a,b);printf("The result is %d\n", product);
                  break;        break;
                default:  printf("Operation not defined\n");
        }
}
```

```
void add(int f, int g);
void subtract(int f, int g);
void multiply(int f, int g);
```

```
/*add() function definition*/
void add(int f, int g)
{
        int sum;
        sum = f+g;
        printf("The result is %d\n", sum);
}
```

```
/*subtract() function definition*/
void subtract(int f, int g)
{
        int difference;
        difference = f-g;
        printf("The result is %d\n", difference);
}
```

```
/*multiply() function definition*/
void multiply(int f, int g) {
        int product;
        product = f*g;
        printf("The result is %d\n", product);
}
```

Calculator using programmer-defined functions

```
#include <stdio.h>
void add(int f, int g);
void subtract(int f, int g);
void multiply(int f, int g);
void main(void)
{
      char c;
      int a, b;
      printf("Please enter the operation:\n");
      scanf("%c",&c);
      printf("Please enter two integers\n");
      scanf("%d%d",&a,&b);
      switch(c)
      {
            case '+': printf("This is an addition\n");
                      add(a,b);
                      break;
            case '-': printf("This is a subtraction\n");
                      subtract(a,b);
                      break;
            case '*': printf("This is a multiplication\n");
                      multiply(a,b);
                      break;
            default:  printf(" Operation not defined\n");
       }
}
```

```
/*add() function definition*/
void add(int f, int g) {
        int sum;
        sum = f+g;
        printf("The result is %d\n", sum);
}

/*subtract() function definition*/
void subtract(int f, int g) {
        int difference;
        difference = f-g;
        printf("The result is %d\n", difference);
}

/*multiply() function definition*/
void multiply(int f, int g) {
        int product;
        product = f*g;
        printf("The result is %d\n", product);
}
```

Summary of Lectures #15

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

  a function were discussed

- Steps for writing a programmer-defined function

Things To Do

- Review lectures and labs

Next Topic

- Functions (Cont'd)
  - Parameter passing
  - Standard library functions