CMSC 11: FUNCTIONS

by: Maverick C. Crisostomo

Objectives:

At the end of the lesson, you should have:

1. Learned the types of functions:
   a. *Built-in functions* (e.g. math functions); and
   b. Programmer-defined functions;
2. Learned the importance of modular programming.

FUNCTIONS

A function is a block of code that has a name and it has a property that it is ***reusable***. It is a self contained block of statements that ***perform a coherent task*** of same kind.

A subprogram; a program within a program.

“A function should do only one thing”.

SYNTAX:

```
[return-type] function-name ( arguments )
{
        . . . local variable declaration . . . ;

        . . . statements . . . ;

        [return return-value;]

}
```

Function Header

- Return value type (*optional*)
- Name of the function
- Parameters or arguments of the function

Function Body

Whatever is written with in **{ }** in the example is the body of the function.

Note: A good function name is something of the *verbNoun* form (e.g. *computeAverage*).

EXAMPLES:

A function that does not return any value		**A function that returns a value**

void printHello(void)
{
printf(“hello”);
}
	printHello()
	{
	printf(“hello”);
	}
		int computeSum(int x, int y)
		{
		int intSum;
		intSum = x + y;
		return intSum;
		}

Note: Not all functions need to take arguments or return a value. *void* in the above example indicates that a function does not return anything. Also, if a function does not take any parameter(s), its parameter list is empty, it can contain the keyword *void* but that style is now out favor.

IMPLEMENTING A FUNCTION

A program without function declaration/prototype

```
#include <stdio.h>


/* This function computes the double of a number */
int computeTwice (int intNum)
{
   int intResult = intNum * 2;
   return intResult;
}

```

Function Definition

```
main( )
{
    int intNum = 13;
    int intA = 1;
    int intB = 2;
    intA = computeTwice(intA);
    intB = computeTwice(intB + intNum);
}
```

Function Call

A program with function declaration/prototype

```
#include <stdio.h>
int computeTwice(int );
main( )
{
    int intNum = 13;
    int intA = 1;
    int intB = 2;
    intA = computeTwice(intA);
    intB = computeTwice(intB + intNum);
}

/* This function computes the double of a number */
int computeTwice (int intNum)
{
   int intResult = intNum * 2;
   return intResult;
}
```

Function Declaration / Prototype

Function Call

Function Definition

Things to notice…

- The expression passed to a function by its caller is called the ***actual parameter***－such as “a” and “intB + intNum” (see Line 9). The parameter local to the function is called the ***formal parameter***－such as the “intNum” in “int computeTwice(int intNum)” (see Line 13).
- Parameters are passed “***by value”***. The value in the ***actual parameter*** is copied into the function’s ***formal parameter*** just before the function begins executing. So, any modification to the arguments does not affect the values of the variables used in the function call.
- The variables local to the function *computeTwice()*, **intNum** and **intResult**, only **exist temporarily while computeTwice() is executing.** In the same way, as variables local to main and other functions exist only when function **main()** is executing.
- ***Function declarations/prototypes*** go before the main function.
- ***Function definitions*** go after the main function.

LOCAL VARIABLES vs. GLOBAL VARIABLES

Local Variables	Global Variables
• Parameters and variables declared inside the definition of a function. • Exist only inside the function body. • Once the function returns, the variables no longer exist!	• Variables declared outside any of the function definition. • Any function can access/change global variables.

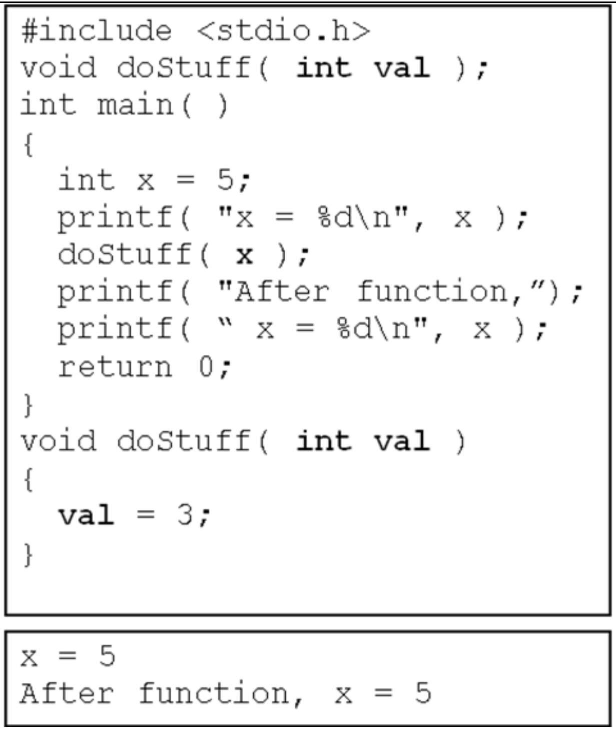
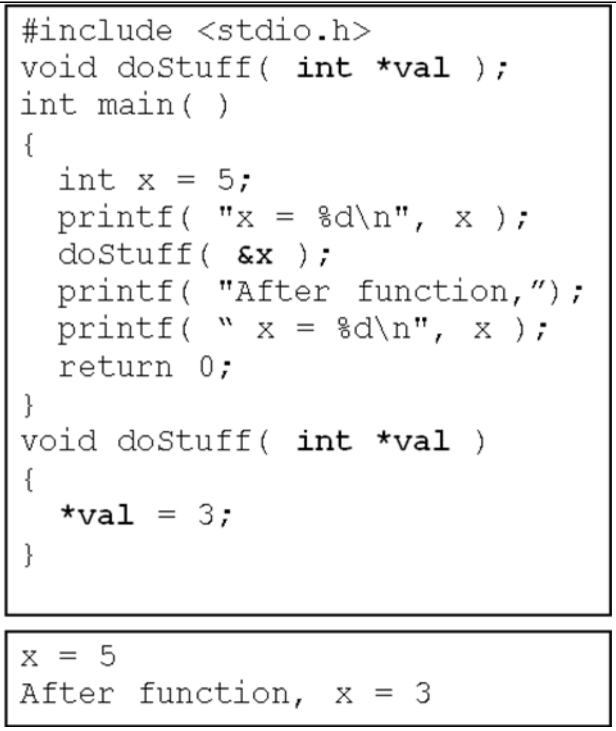
Note: Global variables become hard to manage when you begin to work with multiple functions. Thus, it is a good practice to declare variables only in the scope where they are needed.

PASSED-BY-VALUE vs. PASSED-BY-REFERENCE

PASSED-BY-VALUE	PASSED-BY-REFERENCE
• All arguments are copied, and the copies are passed into the function. • Any modification to the arguments does not affect the values of the variables used	• Passing the address of the variables instead of passing the value in that address. • If we want a module to affect the variables in the calling function, we can send the

in the function call.	*address* of the variables.

EXAMPLE:

Pass by VALUE	Pass by REFERENCE
	

Pass by VALUE:

```
#include <stdio.h>
void doStuff( int val );
int main( )
{
  int x = 5;
  printf( "x = %d\n", x );
  doStuff( x );
  printf( "After function,”);
  printf( “ x = %d\n", x );
  return 0;
}
void doStuff( int val )
{
  val = 3;
}
```

```
x = 5
After function, x = 5
```

Pass by REFERENCE:

```
#include <stdio.h>
void doStuff( int *val );
int main( )
{
  int x = 5;
  printf( "x = %d\n", x );
  doStuff( &x );
  printf( "After function,”);
  printf( “ x = %d\n", x );
  return 0;
}
void doStuff( int *val )
{
  *val = 3;
}
```

```
x = 5
After function, x = 3
```

BUILT-IN FUNCTIONS/LIBRARY FUNCTIONS IN C

- Functions that are used so frequently that they are provided for you by the compiler.
- These functions are declared in the appropriate header files
- Library Function Examples
  - Mathematics functions
    - pow, sqrt **–** found in **cmath.h**
    - abs, rand – found in **stdlib.h**
    - floor, fmod, - found in **math.h**
  - Trigonometric functions

        - sin, cos, tan, acos, asin, atan – also found in **cmath.h**
    - String manipulation functions
        - strcat, strcpy – found in **cstring.h**
    - Standard Input functions
        - scanf, printf, getc, getchar, putc, putchar – found in **stdio.h**
    - Other functions
        - exit – found in **stdlib.h**

IMPORTANCE OF MODULAR PROGRAMMING

- A function is a module which is written to define one well-defined task.
- By breaking a program into modules, a programmer is able to:
    - test the modules independently thus, making it easier to test, debug and correct the whole program.
    - easily read and understand the program
    - reuse the functions and reduce recurrences in the program
    - easily maintain the whole program.

SOME NOTES

- Any function can be called from any other function even main ( ) can be called from other functions.
- A function can be called any number of times.
- The order in which the functions are defined in a program and the order in which they get called need not necessarily be same.
- A function can call itself such a process as called ‘recursion’.
- Any C program contains at least one function. If a program contains only one function, it must be main( ).

- In a C program if there are more than one function present then one of these function must be main( ) because program execution always begins with main( ).

- There is no limit on the number of functions that might be present in a C program.

- Each function in a program is called in the sequence specified by the function calls in main( ).

- After each function has done its thing, control returns to the main( ), when main( ) runs out of function calls, the program ends.

- Any function by default returns an int value.

Practice Exercises:

1. Write a function named maxOf2 that takes two integer arguments and returns the larger of the two.
2. Write function named maxOf3 that takes three integer arguments and returns the largest of the three, making use of your function maxOf2 that you previously created.

3. Write a function celsius() to convert degrees Fahrenheit to degrees Celsius. (The conversion formula is °C = 5/9 * (°F - 32). Write a function fahrenheit() to convert degrees Celsius to degrees Fahrenheit. (The conversion formula is °F =(9/5 * C) + 32).