CSC 201 CSC 150
C++ Programming

Dr. Mazen Zainedin

Stat & OR Dept.

College of sciences KSU

Lecture 4: **Functions**

- **<u>Functions as modules:</u>**
- A function is a block of code with a name.
- A function is said to be “invoked” or “called” via a “function call” from “calling function” or “caller” (in this case, main). To call a function, type the name of the function, followed by parentheses.
- Now that we’ve packaged up each part of our big procedure and given it a name.
- We can also now split development of the program among multiple people, with each developing a different set of functions. This sort of independence of program components is called *modularity, and is critical to good software engineering.*

Function Definitions:

Example:

```
void printHello()
 {
    cout << "Hello world!";
 }
```

This definition specifies that we want to name the sequence of commands within the curly braces ({…}) printHello, so that we can then call it from another function, such as main, with the syntax printHello();

Example:

```
bool isMultiple(int a, int b)
 {
    if( a% b == 0 )
        return true;
    else
        return false;
 }
```

Here, bool is the *return type, isMultiple is the function name, a and b are argument names, and return true; and return false; are return statements.*

Parameters/Arguments:

In the isMultiple function definition,
a and b are arguments – variables that are declared between the parentheses of the function definition and can then be used throughout the function body.

The function call isMultiple(25, 5) specifies that when the body of the isMultiple function is executed for this call, the variables a and b in the function should store the numbers 25 and 5, respectively.

C++ functions in can be compared to mathematical functions: just like f(x, y, z)

Function Prototypes:

If you try to call a function before it is defined in your source file, you will get a syntax error. The solution is to know the compiler know ahead of time what's coming with a *function prototype.*

This looks exactly like the first line of a function declaration: return type, function name, parentheses, and argument list. Instead of braces, though, you just put a semicolon. You also don't need to include the names of the variables in the argument list.

Function prototypes are usually put either at the beginning of source files, or in separate *header files (usually with .h extensions) which are then included with the #include preprocessor directive. Usually only large groups of prototypes should be put in header files.*

Example: If you define isPrime after main in your source file but want to call it from main, put the line bool isPrime(int); or the line bool isPrime(int number); before main.

Standard Library Functions:

There are lots of prepackaged functions to use in standard C++ libraries. To use them, you must include the appropriate header files. These functions are very efficient and well-implemented. They should be used in preference to handcrafted functions whenever possible.

Example: The sqrt function (in header file cmath) takes square roots. The rand function (in cstdlib) generates random integers.

Scope:

Variables exist within *scopes. An identifier can be referenced anywhere within its scope, as long as the reference comes after its declaration.*

Global variables – variables declared outside of any function – have file scope, meaning they can be referred to from anywhere in the file. Global variables should generally be avoided, except for global named constants.

Every set of braces is its own scope, and can contain *local variables. The scope of arguments to a function is the entire function body.*