Function Basics

1. A function must have a name.

2. Naming rules are the same as for other C++ identifiers: must begin with a letter or underscore; can contain letters, numbers, and underscores (and **nothing** else); must be unique (can't be a reserved word or declared previously) and are case-dependent.

   A common practice (which we will use in this class) is use a combination of upper- and lower-case letters, capitalizing each "word" in the name including the first. Numbers may be included, if appropriate, but are rarely used.

3. There must be a set of parentheses immediately after the name; they may or may not contain something.

4. The header of a function contains the return type followed by the function name and the parentheses. The body of the function consists of program statements enclosed in braces and immediately follows the function header. The complete function consists of the function header and the body. The code for a complete function is also referred to as the **function definition**.

5. In a function definition, the header is **not** followed by a semicolon.

6. A function call is an executable statement that causes the function code to be executed. A function call is always followed by a semicolon.

7. Functions may call other functions (and be called by other functions). In each case, control is transferred to the called function and when it is finished, control returns to the calling function.

8. All functions are capable of returning a value. To return a value to the calling function, use the "return" statement followed by a value.

9. In order to use a returned value, the function call must be part of a larger statement. The most common way to call a value-returning function is to call the function on the right side of an assignment statement. Other ways include putting it in an output statement or as part of an if or loop condition.

10. The type of value a function returns is said to be the function's type. A function may return any C++ data type including user-defined types. A function of type int can return only a value of type int,a function of type float can return only a value of type float, etc. A function of type void returns no value.

11. In a function header, the parentheses may contain one or more local variable declarations. These declarations are called **parameters** and are separated from each other by commas. Parameters receive values passed by the function call.

12. The function header always begins with the return type. The header

    ```
    float ConvertToFahrenheit(int cTemp)
    ```

    indicates that the function receives a parameter of type float and returns a value of type int.

13. The calling statement never contains data types. Values passed to the function in the parentheses are called arguments. A calling statement for the above header could be

    ```
    fahrenheitTemp = ConvertToFahrenheit(30);
    ```

    or

    ```
    cout << ConvertToFahrenheit(30);
    ```

14. Ansi C++ requires that any user-defined names must be declared in the program before they are used in a program statement. This is usually done by writing a function declaration or **prototype** near the beginning of the program. A prototype is just a copy of the function header followed by a semicolon. The prototype guarantees that the function receives the right number and type of values and returns the right type of value.

Scope of Variables

Declarations in C++ may be global or local. At this time we are primarily concerned with declarations of variables, but you should be aware the scope rules apply to any user-defined objects including constants, data types, and function prototypes:

- Global variables are active throughout a program and are known (declared) in all functions.
- Local variables are declared within a function or block. They are recognized only in that part of the program. They are destroyed and no longer defined when the function or block ends.

Features:

- Global variables are declared outside a routine, usually at the top of the program. Local variables are defined within a routine or block.
- There are several important exceptions to the rule that identifier names must be unique. A local variable may have the same name as a global variable. In addition, you can use the same name for multiple local variables as long as they are declared in different functions or blocks. Also, in C++ but not in C, multiple functions may be written with the same name so long as they contain different numbers or types of parameters. The name for this is **function overloading**.
- If a function or block contains a local variable that has the same name as a global variable, the global variable is **hidden** and cannot be accessed. This is also true of local variables defined inside and outside of a program block.
- Global variables can be hidden from other parts of the program as well. If a global variable is defined between functions, it is invisible to all functions written above it.
- Global variables are dangerous and can cause unwanted side effects. Generally speaking, you should avoid using global variables.