***C++ Basics (Chapter 2)***

**Functions –**

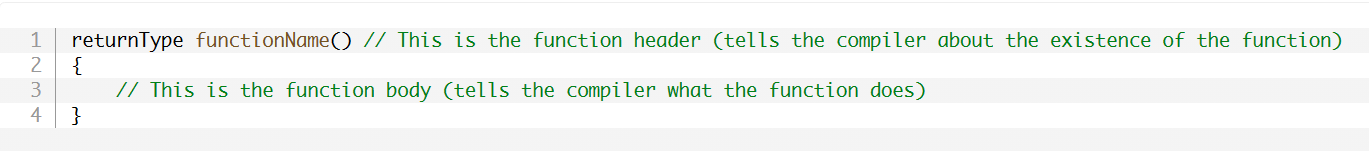
A **function**is a *reusable* sequence of statements designed to do a particular job.

Functions that you write yourself are called user-defined functions.

A **function call** is an expression that tells the CPU to interrupt the current function and execute another function to perform a task defined in the program. The *CPU* puts a ‘bookmark’ at the current point of execution, and **calls (**executes**)** the function named in the function call. When the *called* function ends, the CPU returns to the bookmarked point in the execution.

The function initiating the call is the **caller** and the function being called is the **called** (callee) function.

**User-defined function**



The first line (returnType functionName()) is called the **function header**, and it tells the compiler about the existence of a function, it’s name, and some other information (like return type and parameter type).

* The functionName is the name (identifier) of the *user-defined* function.
* The **parentheses** after the *identifier* tell the compiler we’re defining a function.

The curly braces and the statements inside are called the **function body**. This is where the statements go that determine the function’s purpose.

*Example –* Shows function *definition* and *call* –

A screenshot of a computer code

Description automatically generated

The program outputs –

A white background with black text

Description automatically generated

* The program begins execution at the top, the first line to be executed prints “Starting main()”.
* The second line calls the void function (doPrint()) defined above.

**Warning:** Don’t forget to include the parentheses () after the function’s name when making a function call.

Because a function call was made, execution of statements in *main* is suspended, and execution jumps to the top of the function doPrint(). The first (and only) line defined in doPrint prints “In doPrint()”. When doPrint terminates the program returns back to the bookmarked point in execution and prints the next line defined in *main* which prints “Ending main()”.

**Calling functions more than once –**

A screenshot of a computer program

Description automatically generated

The program produces the output –

A white background with black text

Description automatically generated

* doPrint is called twice by main, so it executes twice, and “in doPrint()” gets printed twice (once for each call).

**Functions that call other functions –**

Any function can call any other function. In the following program, function *main* calls function doA, which calls function doB.

A screenshot of a computer program

Description automatically generated

Output –

A white background with black text

Description automatically generated

**Nested functions are NOT supported –**

Unlike other programming languages, in C++, functions cannot be defined inside other functions.

The following program is not legal –

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generatedThe proper way to write it would be –

**Quiz.**

1. In a function definition, what are the curly braces and statements in-between called?

* The **function body**.

1. What does the following program print?

A screenshot of a computer program

Description automatically generated

The program prints –

**“Starting main()”  
“In doA()”  
“In doB()”  
“Ending main()”**

**X.**

“Starting main()”  
“In doA()”  
“In doB()” - **Read the code *carefully***  
“In doB()”  
“Ending main()”

**Function return values (value-returning functions)**

A screenshot of a computer code

Description automatically generated

The program is simple, it takes a value from a user then returns the value doubled.

The program does not need to be broken up (multiple functions) but if it were to be, how would you go about it?

*This is an attempt* (**incorrect**) –

A screenshot of a computer code

Description automatically generated

* The program doesn’t work because the value for num{} is never initialized after receiving input from user with the getValueFromUser() function.

The program needs some way for getValueFromUser to return the value back to *main* so that *main* can make user of the data.

**Return Values –**

To return a value back to the *caller,* two things are needed.

First,

* Function must indicate what type of value it will return; this is the functions **return type.**
* Function ‘getValueFromUser’ has a return type *void* (meaning no value is returned), function ‘main’ has a return value of *int*.
  + Note: This doesn’t determine what *specific* value is returned – it only determines the *type* of value that will be returned by the function.

Second,

* We use a **return statement** to indicate the specific value being returned to the caller. The specific value returned is called the **return value.** When the *return statement* is executed the function exits immediately, and the return value is copied from the function to the *caller* (main() in this case).
* This process is called ***return by value.***

*Sample program that returns an integer value to a* ***caller*** *–*

*A screenshot of a computer

Description automatically generated*

The program prints –

A white rectangular object with a black border

Description automatically generated

**Fixing the previous problem –**

**A screenshot of a computer program

Description automatically generated**

The program initializes *int num*, it sees that there’s a function call to getValueFromUser(), it executes the function which requests the user to input a number and returns it back to the *main* function. The main function then receives the **return value** from getValueFromUser which is used as the initialization value for variable *num*.

**Revisiting main() –**

When the program is executed, the operating system makes a *function call* to main. Execution then jumps to the top of main. The statements in *main* are executed sequentially. Finally, main returns an integer value (usually 0) and the program terminate. The return value from *main* is called a **status code** (also sometimes, an *exit code,* or *return code).* It is used to indicate whether a program ran successfully or not.

**Best Practice:** Your main function should return the value 0 if the program ran normally, a non-zero value will result in *undefined behavior.*

**A value-returning function that does not return a value will produce undefined behavior –**

A function that returns a value is called a **value-returning function.** A function is value-returning if the return type is anything than *void.*

A value-returning function *must* return a value of that type (using a return statement), otherwise undefined behavior will result.

*Example program with undefined behavior –*

A screenshot of a computer code

Description automatically generated

* Note: A modern compiler should generate a warning because getValueFromUserUB is defined as returning an int type but has no return statement. Running this program will result in *undefined behaviour,* because getValueFromUserUB is a value-returning function with no *return statement.*

**Best practice:** Make sure non-void functions return a value in all cases. Failure to do so will result in *undefined behaviour.*

**Function main** – will *implicitly* return value 0 if no return statement is provided. That said, it is best practice to include a return statement to show your *intent* and keep consistency with the other functions (which will result in undefined behaviour without a return value).

**Functions can only return a *single value –***

A value-returning function can only return a single value each time it is called.

Note that the value returned does not need to be literal – it can be the result of a valid expression, including a variable or even a call to another function that returns a value. In the getValueFromUser example *above,* we returned a variable ‘input’, which held the user input.

**The function author can decide what the return value means –**

The meaning of a function is determined by the function’s author. Some functions are used to return a calculated value, some use return values as status codes (failure or success), other functions return nothing (void functions).

* Because of the wide variety of possibilities, it’s *best practice* to indicate what the return values mean:

A screenshot of a computer code

Description automatically generated

**Reusing functions –**

*Consider the following program –*

A screenshot of a computer code

Description automatically generated

While the program works, it’s a little redundant (repeats itself which is not good practice).

**DRY =** “Don’t Repeat Yourself’.

The program *improved –*

A computer screen with text on it

Description automatically generated

Outputs:

A black background with white text

Description automatically generated

In the program we call getValueFromUser twice, to initialize variables x and y. This saves us from duplicating code and reduces the odds of making a mistake.

This is the essence of *modular programming*; the ability to write a function, test it, ensure that it works, and then know that we can reuse it as many times as we want, and it will continue to work.

**Best Practice:** Follow DRY: “Don’t Repeat Yourself”. If you need to do something more than once, consider how to modify the code to remove as much redundancy as possible (the possibility of mistakes). Like all best practice, DRY is meant to be a *guideline,* not an absolute. DRY can harm overall comprehension when code is broken into pieces that are too small.

**Quiz.**

Inspect each of the following programs. Determine what the program will output, or whether the program will generate a compilation error.

1. A screenshot of a computer code

   Description automatically generated

The program will output: “**16”.**

1. A screenshot of a computer code

   Description automatically generated

This program will generate a compilation error because a function cannot be defined inside another function. (**Nested** **functions).**

1. A screenshot of a computer code

   Description automatically generated

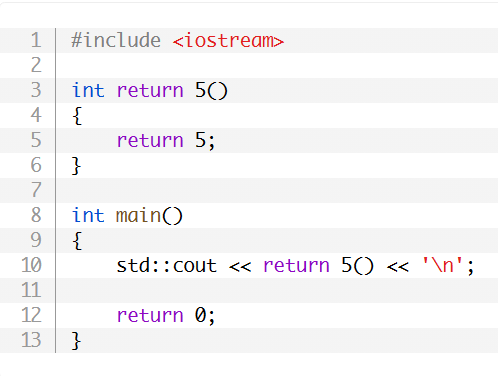
The program will compile correctly but there will be no output.

1. A screenshot of a computer code

   Description automatically generated

Incorrect, there can only be one return statement per function. Because:

(The program prints 5 twice. Both times when getNumbers is called, the value 5 is returned, when the return 5 statement is executed, the function immediately exits, so the return 7 statement is never executed.)

1. 

The function call is incorrect (X. The function has an invalid name. **READ THE QUESTION).**

1. A screenshot of a computer code

   Description automatically generated

The program doesn’t compile because the function call is incorrect (missing parentheses).

1. What does “DRY” stand for, and why is it a useful practice to follow?

DRY stands for “Don’t Repeat Yourself”. It is a practice that involves writing your code so that it reduces redundancy. This makes the program more *concise, less error prone, and more maintainable.*

**Void Functions (***non-value returning functions) –*

*Void return values:* Functions are not required to return a value back to the caller. To define a function that does not return a value, the return type **void** is used. For example:

A screenshot of a computer code

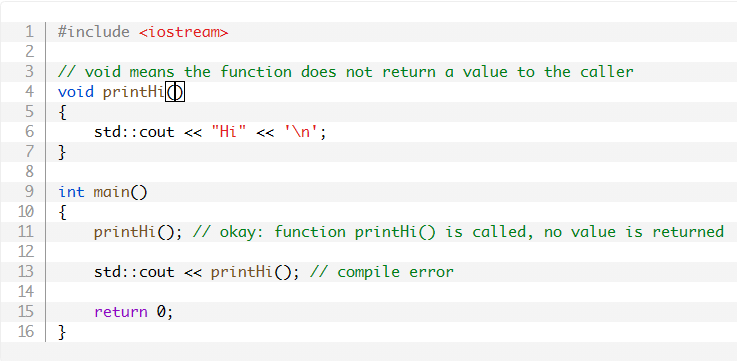
Description automatically generated

In the example provided, printHi is called in *main,* it performs its function (Prints “Hi”), since it’s a void there’s no return statement so the program finishes executing *main* and returns 0.

* *A void function* will automatically return to the caller at the end of the function. No return statement is needed.

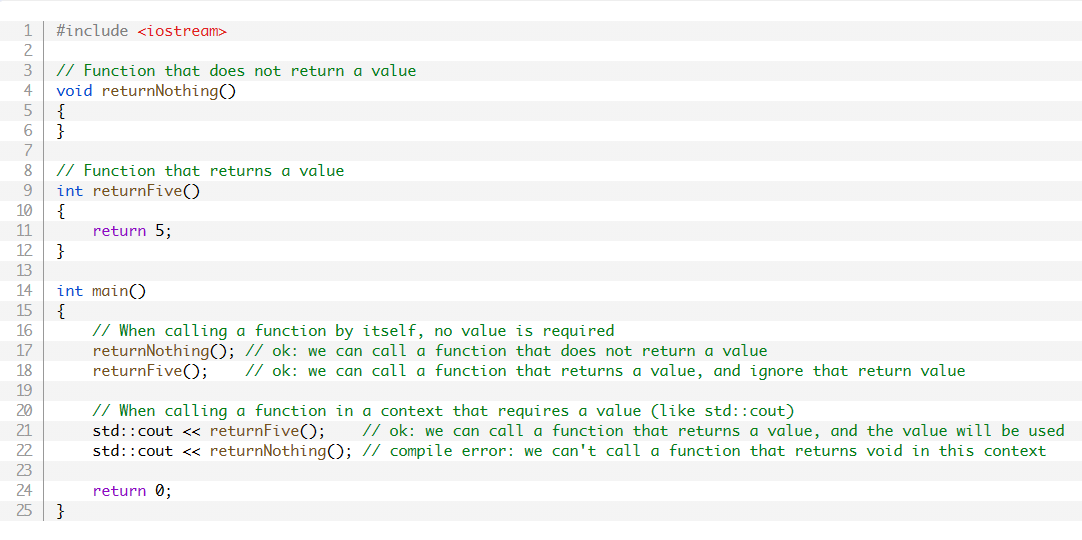
**Best Practice:** Do not put a return statement at the end of a void function.

Void functions **can’t** be used in *expressions* that require a value.



The first call to printHi() is called in a way that doesn’t require a value, so it’s fine.

The second call to printHi() won’t compile because statement is asking to return the value of printHi to ‘std::cout’ which won’t work because the void *printHi()* does not a return value.



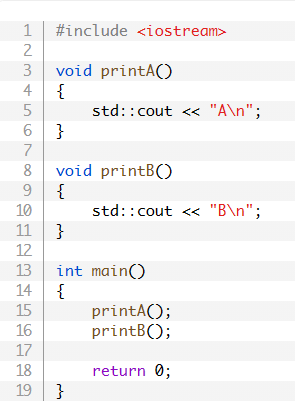
*Returning a value from a void function is a compile* *error.*

A close-up of a computer code

Description automatically generated

**Quiz.**

**Inspect the programs and state the output, or whether they will not compile.**

1. ****

This outputs:

A  
B

1. **A screenshot of a computer code

   Description automatically generated**

This won’t compile since the printA is being called to return a value for cout, which it can’t do because it’s a **void function.**