2.3 — Void functions (non-value returning functions)

learncpp.com/cpp-tutorial/void-functions-non-value-returning-functions/

In a prior lesson (2.1 -- Introduction to functions), we indicated that the syntax for a function definition looks like this:

```
returnType identifier() // identifier replaced with the name of your function
{
// Your code here
}
```

Although we showed examples of functions that had return-type `void`, we did not discuss what this meant. In this lesson, we'll explore functions with a return type of `void`.

Void return values

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

```
#include <iostream>

// void means the function does not return a value to the caller
void printHi()
{
    std::cout << "Hi" << '\n';

    // This function does not return a value so no return statement is needed
}

int main()
{
    printHi(); // okay: function printHi() is called, no value is returned

    return 0;
}
```

In the above example, the `printHi` function has a useful behavior (it prints "Hi") but it doesn't need to return anything back to the caller. Therefore, `printHi` is given a `void` return type.

When `main` calls `printHi`, the code in `printHi` executes, and "Hi" is printed. At the end of `printHi`, control returns to `main` and the program proceeds.

A function that does not return a value is called a **non-value returning function** (or a **void function**).

Void functions don't need a return statement

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

A return statement (with no return value) can be used in a void function -- such a statement will cause the function to return to the caller at the point where the return statement is executed. This is the same thing that happens at the end of the function anyway. Consequently, putting an empty return statement at the end of a void function is redundant:

```
#include <iostream>

// void means the function does not return a value to the caller
void printHi()
{
    std::cout << "Hi" << '\n';

    return; // tell compiler to return to the caller -- this is redundant since the
return will happen at the end of the function anyway!
} // function will return to caller here

int main()
{
    printHi();

    return 0;
}
```

Best practice

Do not put a return statement at the end of a non-value returning function.

Void functions can’t be used in expression that require a value

Some types of expressions require values. For example:

```
#include <iostream>

int main()
{
    std::cout << 5; // ok: 5 is a literal value that we're sending to the console to
be printed
    std::cout << ;  // compile error: no value provided

    return 0;
}
```

In the above program, the value to be printed needs to be provided on the right-side of the `std::cout <<`. If no value is provided, the compiler will produce a syntax error. Since the second call to `std::cout` does not provide a value to be printed, this causes an error.

Now consider the following program:

```
#include <iostream>

// void means the function does not return a value to the caller
void printHi()
{
    std::cout << "Hi" << '\n';
}

int main()
{
    printHi(); // okay: function printHi() is called, no value is returned

    std::cout << printHi(); // compile error

    return 0;
}
```

The first call to `printHi()` is called in a context that does not require a value. Since the function doesn't return a value, this is fine.

The second function call to function `printHi()` won't even compile. Function `printHi` has a `void` return type, meaning it doesn't return a value. However, this statement is trying to send the return value of `printHi` to `std::cout` to be printed. `std::cout` doesn't know how to handle this (what value would it output?). Consequently, the compiler will flag this as an error. You'll need to comment out this line of code in order to make your code compile.

Tip

Some statements require values to be provided, and others don't.

When we have a statement that consists of just a function call (e.g. the first `printHi()` in the above example), we're calling a function for its behavior, not its return value. In this case, we can call either a non-value returning function, or we can call a value-returning function and just ignore the return value.

When we call a function in a context that requires a value (e.g. `std::cout`), a value must be provided. In such a context, we can only call value-returning functions.

```
#include <iostream>

// Function that does not return a value
void returnNothing()
{
}

// Function that returns a value
int returnFive()
{
    return 5;
}

int main()
{
    // When calling a function by itself, no value is required
    returnNothing(); // ok: we can call a function that does not return a value
    returnFive();    // ok: we can call a function that returns a value, and ignore
that return value

    // When calling a function in a context that requires a value (like std::cout)
    std::cout << returnFive();    // ok: we can call a function that returns a value,
and the value will be used
    std::cout << returnNothing(); // compile error: we can't call a function that
returns void in this context

    return 0;
}
```

Returning a value from a void function is a compile error

Trying to return a value from a non-value returning function will result in a compilation error:

```
void printHi() // This function is non-value returning
{
    std::cout << "In printHi()" << '\n';

    return 5; // compile error: we're trying to return a value
}
```

Quiz time

Question #1

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

1a)

```
#include <iostream>

void printA()
{
    std::cout << "A\n";
}

void printB()
{
    std::cout << "B\n";
}

int main()
{
    printA();
    printB();

    return 0;
}
```

Show Solution

1b)

```
#include <iostream>

void printA()
{
    std::cout << "A\n";
}

int main()
{
    std::cout << printA() << '\n';

    return 0;
}
```

Show Solution