1 Functions

Placing all code inside of main() is unmanageable. To make programs easy to understand, you need to break up, or *decompose*, code into concise functions.

In C++, you first declare a function to make it available for other code to use. If the function is used only inside a particular file, you generally declare and define and the function in the source file. If the function is for use by other modules or files, you generally put the declaration in a header file and the definition in a source file.

NOTE Function declarations are often called ”function protoypes” or ”signatures” to emphasize that they represent how the function can be accessed, but not the code behind it.

A function declaration is shown below.

```
void myFunction(int i, char c);
```

Without an actual definition to match this function declaration, the link stage of the compilation process will fail because code that makes use of the funciton will be calling nonexistent code.

```
void myFunction(int i, char c) {
    std::cout << "the value of i is " << i << std::endl;
    std::cout << "the value of c is " << c << std::endl;
}
```

Elsewhere in the program, you can make calls to myFunction() and pass in arguments for the two parameters.

```
myFunction(8, 'a');
myFunction(someInt, 'b');
myFunction(5, someChar);
```

C++ functions can also *return* a value to the caller.

```
int addNumbers(int number1, int number2) {
     return number1 + number2;
}
```

Called as follows

```
int sum = addNumbers(5, 3);
```

Every function has a local predefined variable __func__ that looks as follows:

```
static const char __func__[] = "function-name";
```

This variable can for example be used for logging purposes:

```
int addNumbers(int number1, int number2)
{
std::cout << "Entering function " << __func__ << std::endl;
return number1 + number2;
}
```

2 Alternative Function Syntax

Since C++11, an alternative function syntax is supported with a *trailing return type*. This new syntax is not much use for ordinary functions, but is very useful in the context of specifying the return type of template functions.

```
auto func(int i) -> int
{
    return i + 2;
}
```

The return type of the function is no longer at the beginning, but at the end of th eline after the arrow. main() can also use this syntax.

3 Function Return Type Deduction

C++14 allows you to ask the compiler to figure out the return type of a function automatically. To make use of this, specify auto as the return type and omit any trailing return type

```
auto divideNumbers(double numerator, double denominator)
{
    if (denominator == 0) {/*...*/}
    return numerator / denominator;
}
```

Compiler deduces the return type based on the expressions used for the return statements. Multiple returns should resolve to the same type.