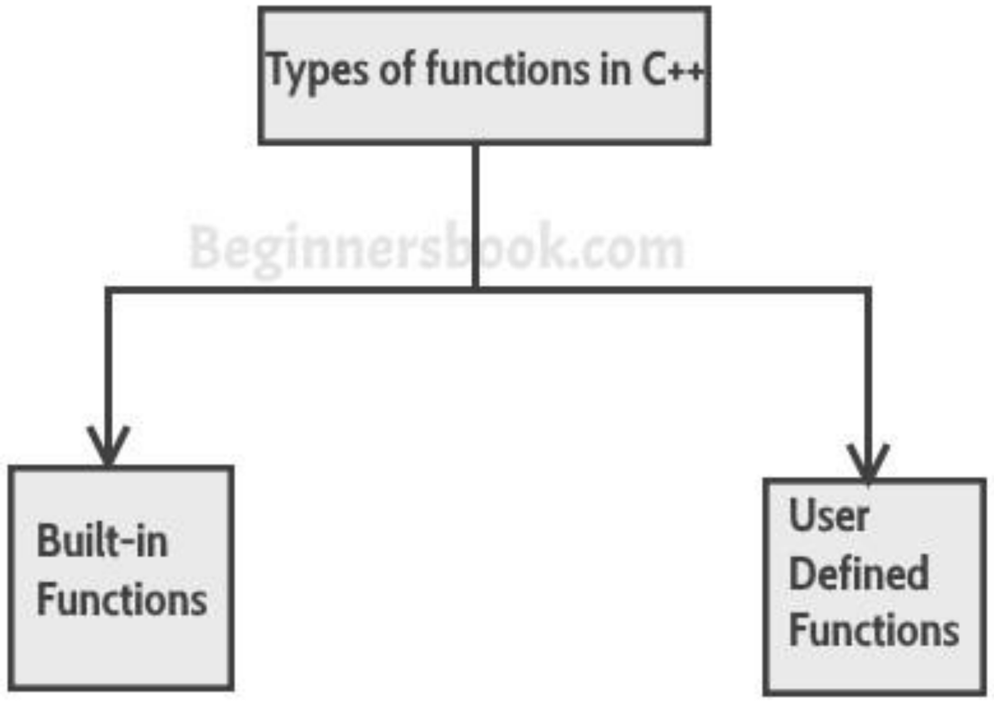
Lecture 11

M M Imran

Function

- A function is a block of code for performing a task.
- A function:
  - Has a name.
  - Has zero or more inputs.
  - May or may not return a value when it completes.
- A function may be defined by:
  - Someone else and made available to an application via an included library.
  - The developer for an application-specific purpose.

Function Types



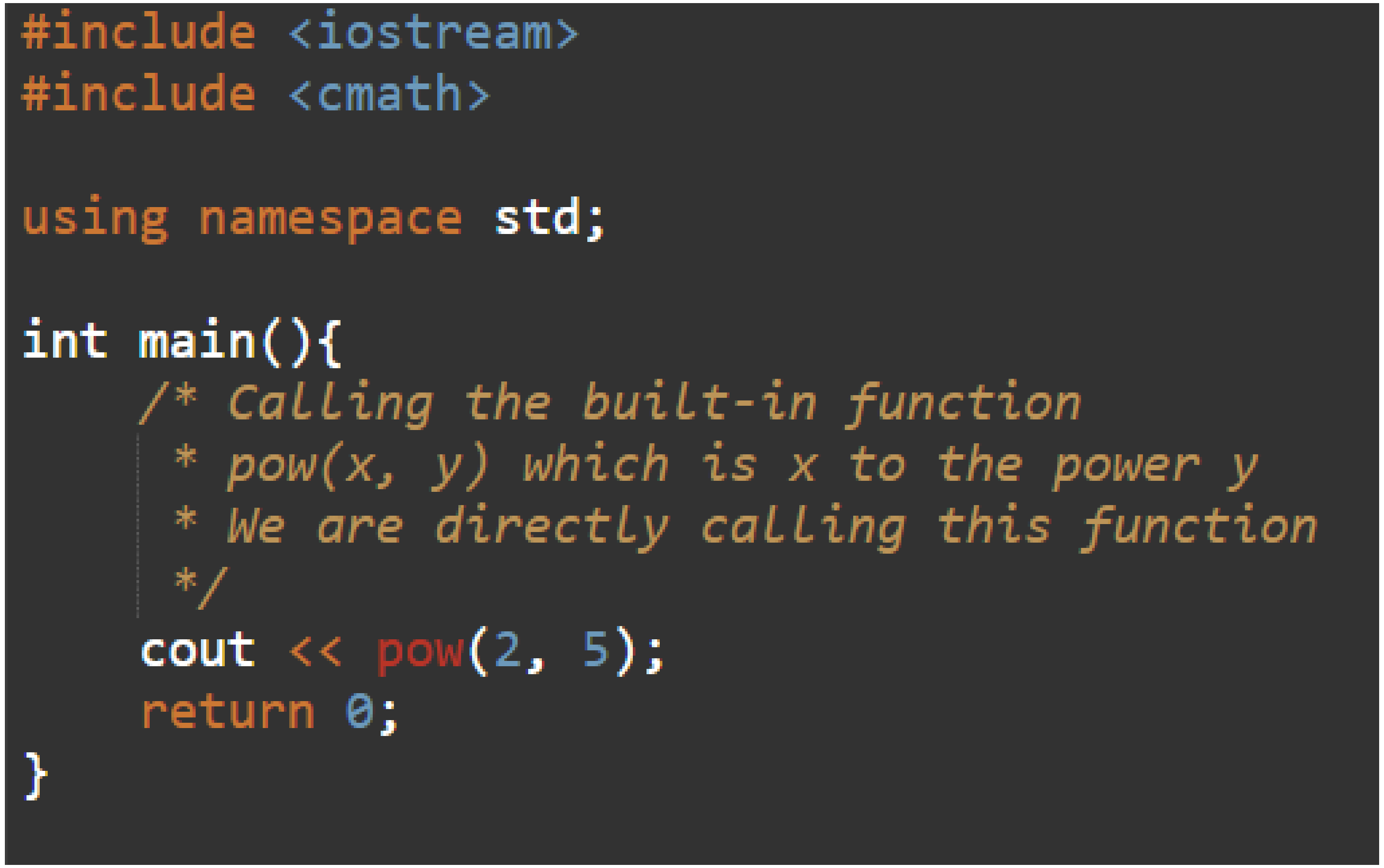
Built-in functions:

- Built-in functions are also known as library functions.
- We need not to declare and define these functions as they are already written in the C++ libraries such as iostream, cmath etc.
- We can directly call them when we need.
- To use pre-defined functions, we include the library containing them.
- There are thousands of functions defined among the C++ libraries.
- See cplusplus.com at www.cplusplus.com/reference/

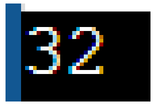
User defined functions:

- C++ allows the programmer to define their own function.
- A user-defined function groups code to perform a specific task and that group of code is given a name (identifier).
- When the function is invoked from any part of the program, it all executes the codes defined in the body of the function.
- A programmer-defined function should be used when there is:
  - A repeated block of code in different parts of an application.
  - A long function that could be divided into two or more functions.

C++ built-in function example



```
#include <iostream>
#include <cmath>

using namespace std;

int main(){
    /* Calling the built-in function
     * pow(x, y) which is x to the power y
     * We are directly calling this function
     */
    cout << pow(2, 5);
    return 0;
}
```



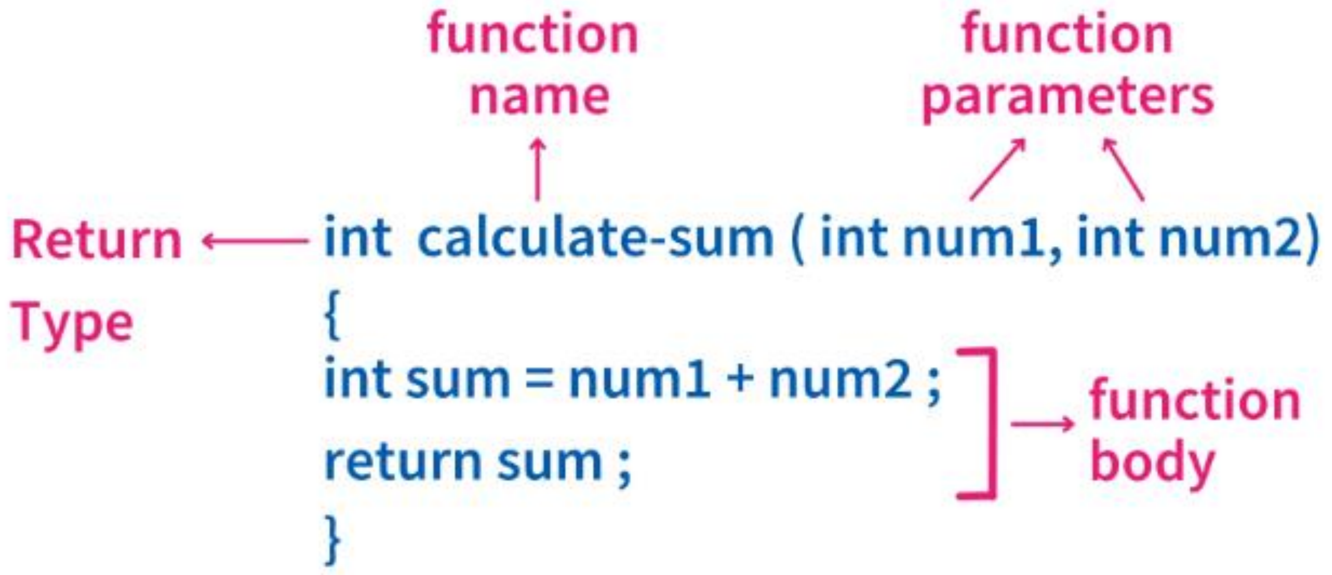
```
32
```

User-defined Functions

- A user-defined function has:
  - Function name
  - Return type
  - Function parameters
  - Function body

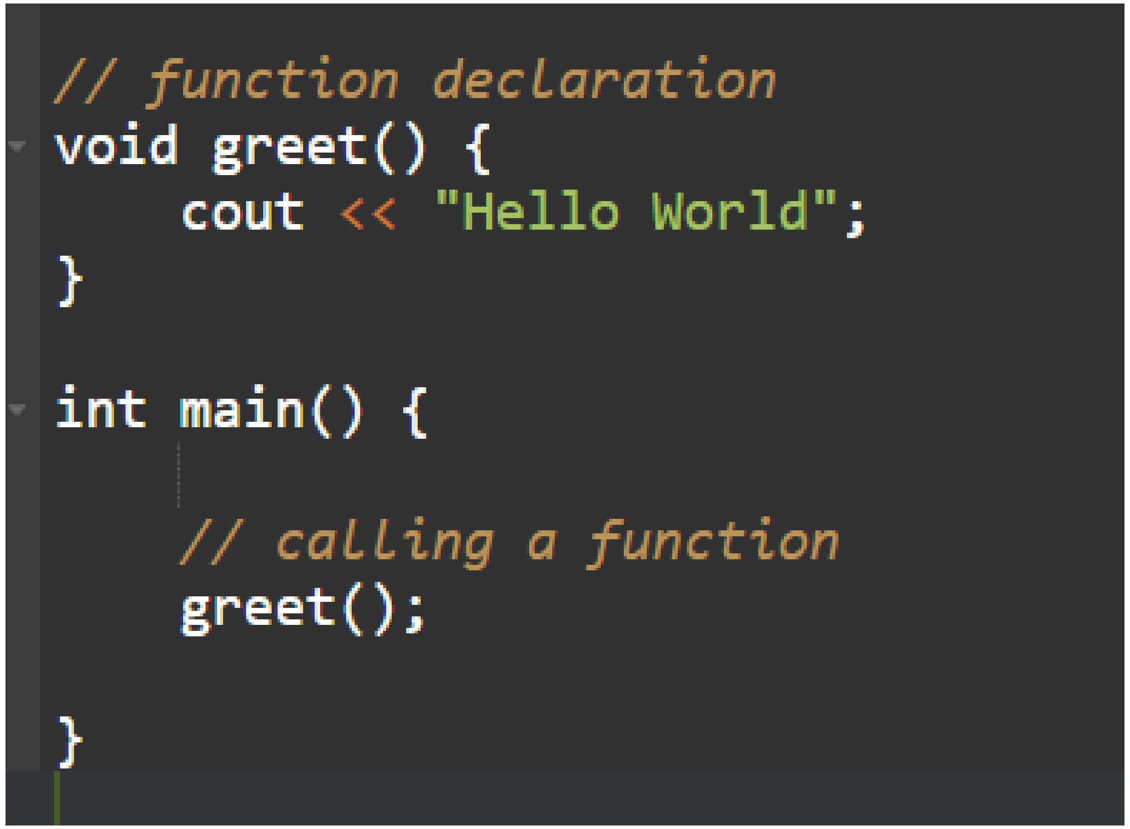
The syntax to declare a function is:

```
returnType functionName (parameter1, parameter2,...) {
    // function body
}
```

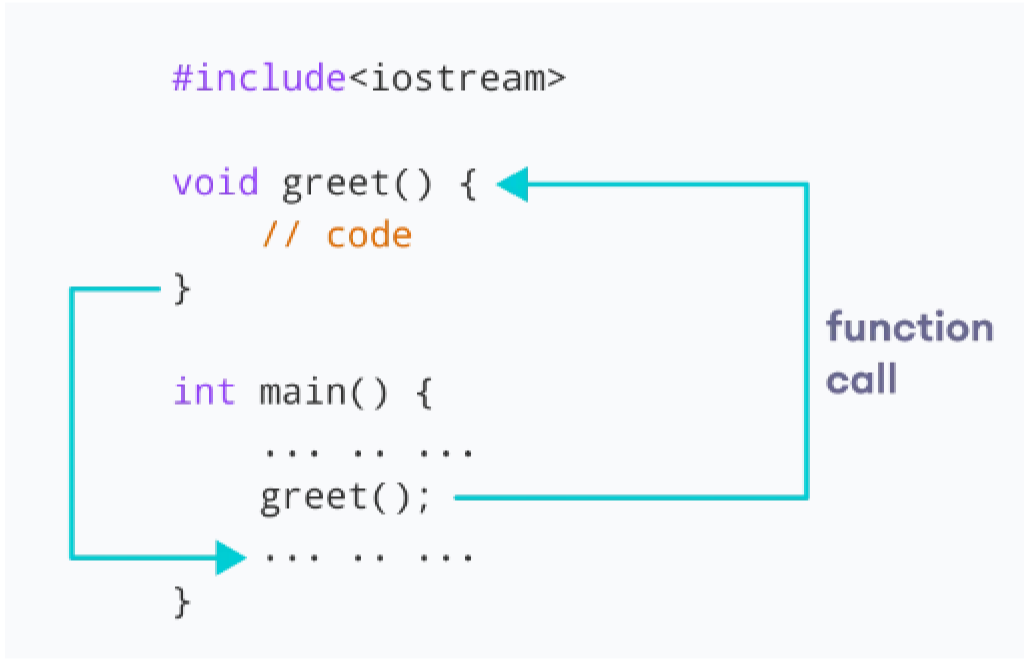


```
int  calculate-sum ( int num1, int num2)
{
int sum = num1 + num2 ;
return sum ;
}
```

Calling a Function



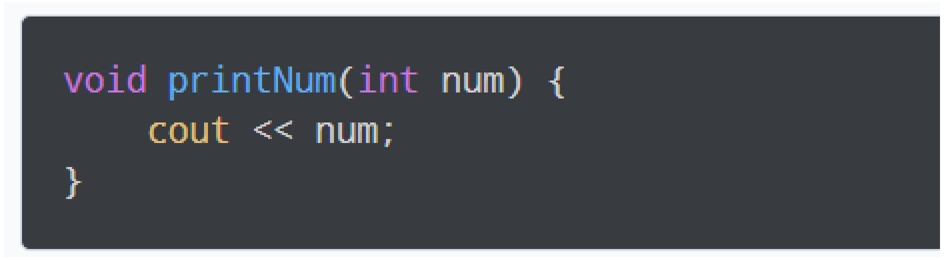
```
// function declaration
void greet() {
    cout << "Hello World";
}

int main() {

    // calling a function
    greet();

}
```



```
#include<iostream>

void greet() {
    // code
}

int main() {
    ... .. ...
    greet();
    ... .. ...
}
```

Function Parameters

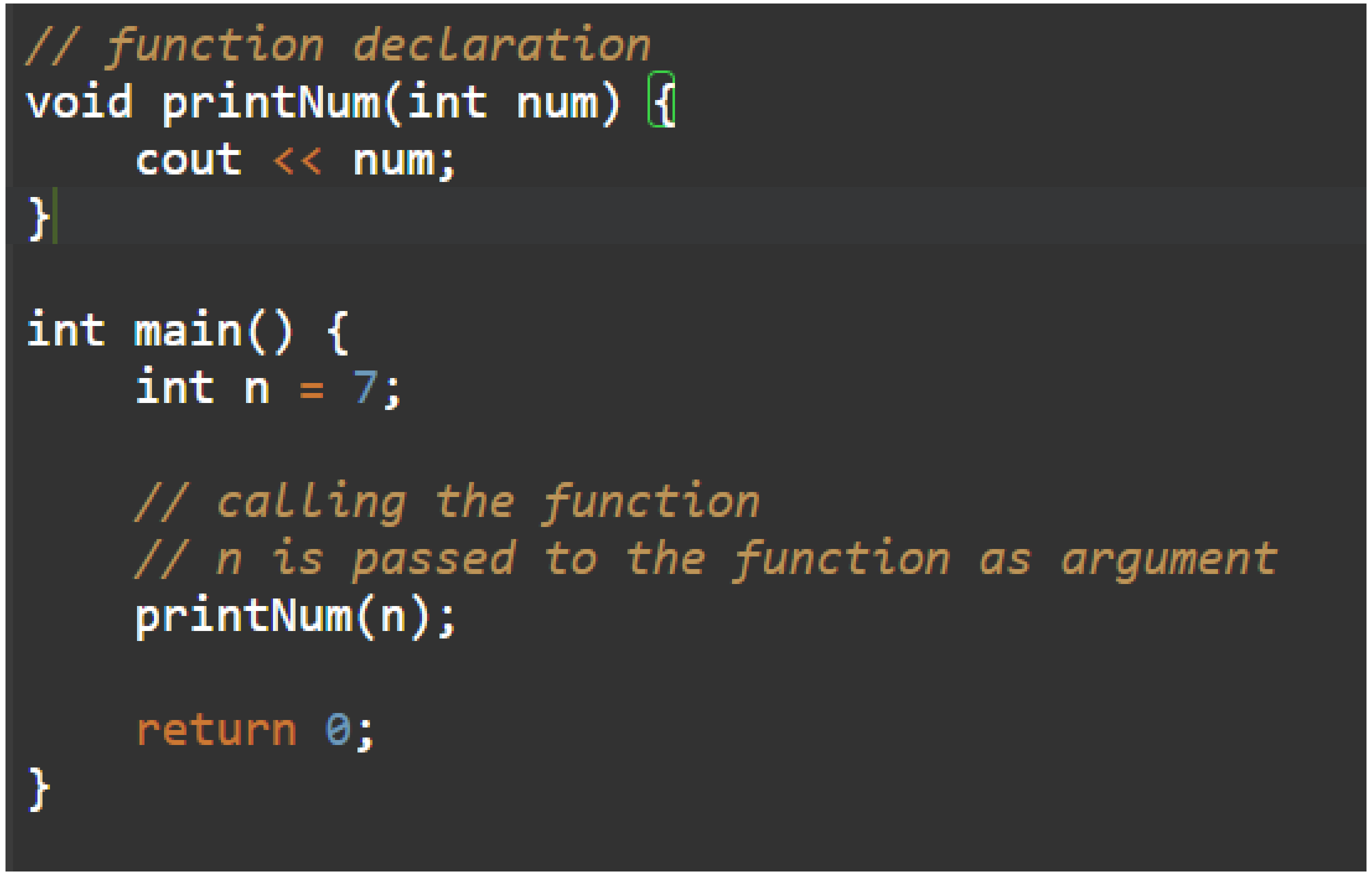
A function can be declared with parameters (arguments). A parameter is a value that is passed when declaring a function.



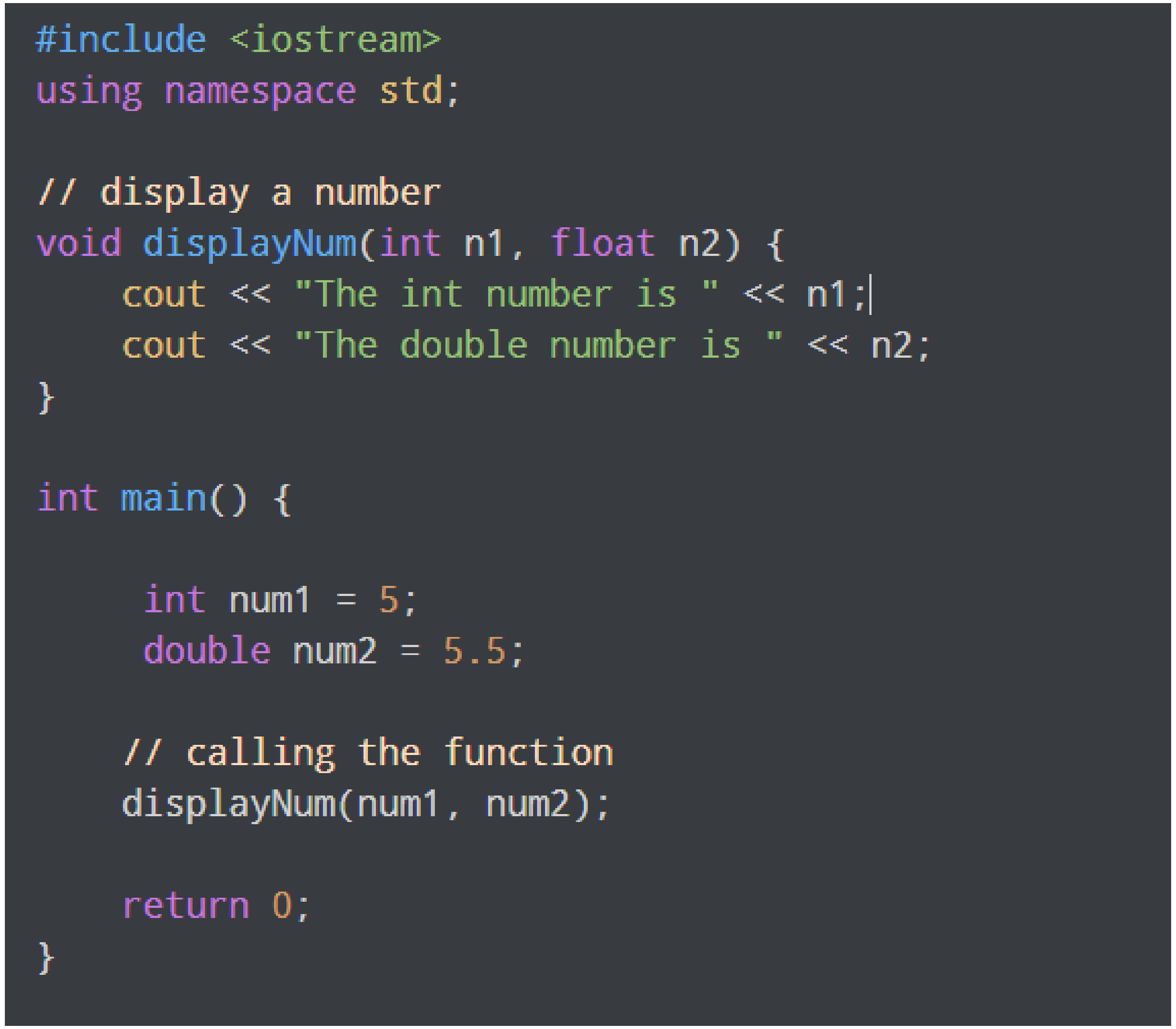
```
void printNum(int num) {
    cout << num;
}
```

Here, the **int** variable **num** is the function parameter.

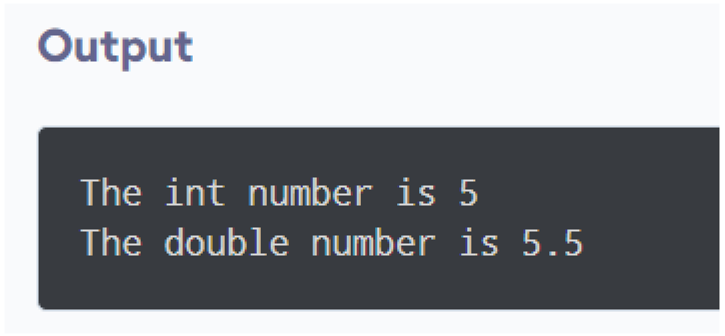
Calling with function parameter



```cpp
// function declaration
void printNum(int num) {
    cout << num;
}

int main() {
    int n = 7;

    // calling the function
    // n is passed to the function as argument
    printNum(n);

    return 0;
}
```



```
#include <iostream>
using namespace std;

// display a number
void displayNum(int n1, float n2) {
    cout << "The int number is " << n1;
    cout << "The double number is " << n2;
}

int main() {

     int num1 = 5;
     double num2 = 5.5;

    // calling the function
    displayNum(num1, num2);

    return 0;
}
```



Output

```
The int number is 5
The double number is 5.5
```

```cpp
#include <iostream>

using namespace std;

// declaring a function
int add(int a, int b) {
    return (a + b);
}

int main() {

    int sum;

    // calling the function and storing
    // the returned value in sum
    sum = add(100, 78);

    cout << "100 + 78 = " << sum << endl;

    return 0;
}
```

Output

```
100 + 78 = 178
```

Void and value function declarations

- Just like a variable, a function must be declared before it can be used.
- A function may or may not return a value when it completes.
- There are two types of functions:
  - A void function runs but does not return a value.
  - A value function runs and then returns a value.

Void function declaration

A void function declaration has syntax:

```
void <function-name>(<parameter-list>)
{
   <block>
}
```

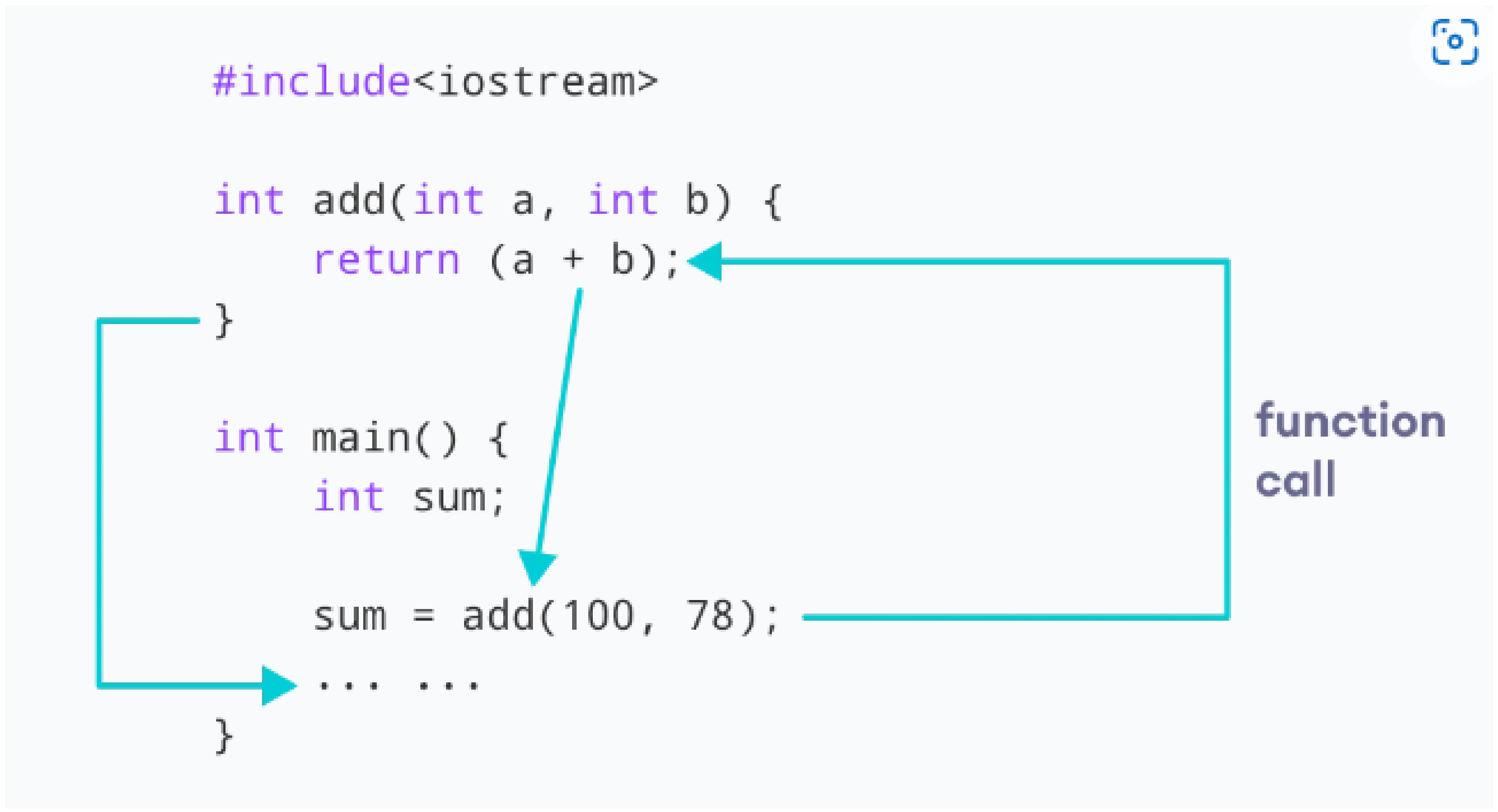
- void means the function does not return a value.
- <parameter-list> is a list of zero or more inputs needed by the function.

Value function declaration

A value function declaration has syntax:

```
<data-type> <function-name>(<parameter-list>)
{
   <block>
   return <expression>;
}
```

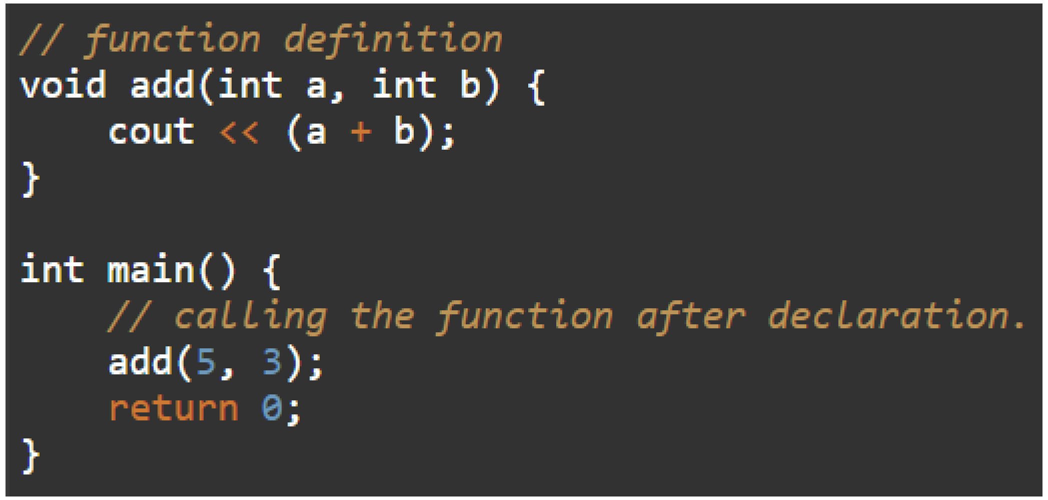
- <data-type> is any primitive or class type.
- <parameter-list> is a list of zero or more inputs needed by the function.
- return <expression>; appears at least once or more times in the function.
- The data type of <expression> must match data type of <data-type>, otherwise a syntax error will result.



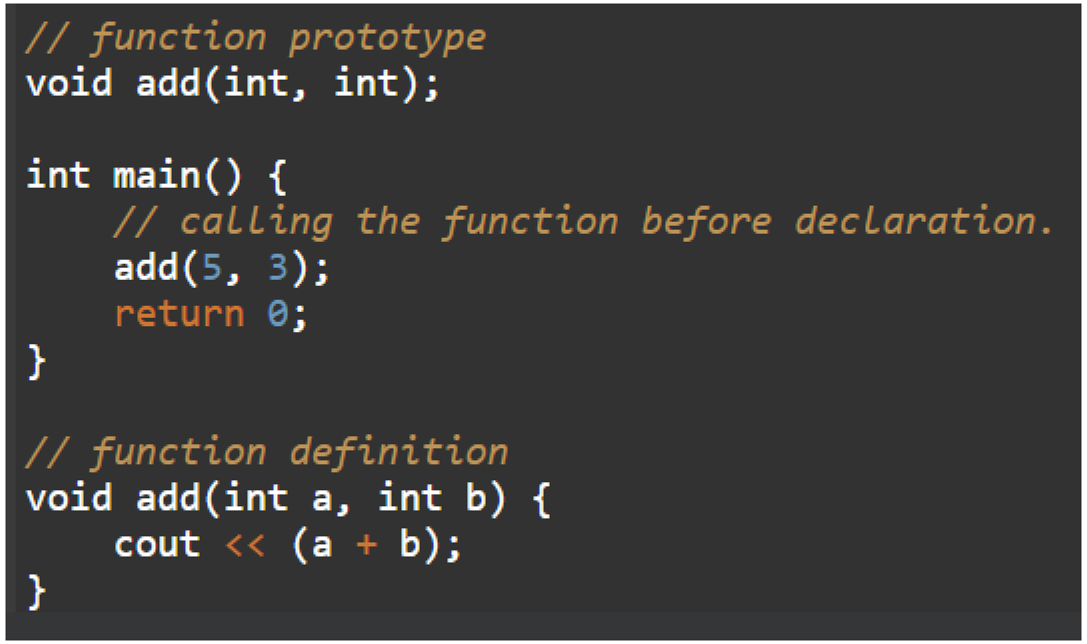
```
#include<iostream>

int add(int a, int b) {
    return (a + b);
}

int main() {
    int sum;

    sum = add(100, 78);
    ... ...
}
```

Function Prototype

- In C++, the code of function declaration should be before the function call.
- However, if we want to define a function after the function call, we need to use the function prototype.



```cpp
// function definition
void add(int a, int b) {
    cout << (a + b);
}

int main() {
    // calling the function after declaration.
    add(5, 3);
    return 0;
}
```



```cpp
// function prototype
void add(int, int);

int main() {
    // calling the function before declaration.
    add(5, 3);
    return 0;
}

// function definition
void add(int a, int b) {
    cout << (a + b);
}
```

Practice

Write a function toThePower(x, y) to find the x^y value.

- Create the function before main function
- Call the **toThePower** function from the **main** function
- Print the result

Practice

- Do the same thing using function prototype.
- **toThePower** function must be after the main function.

Parameters / Formal parameters

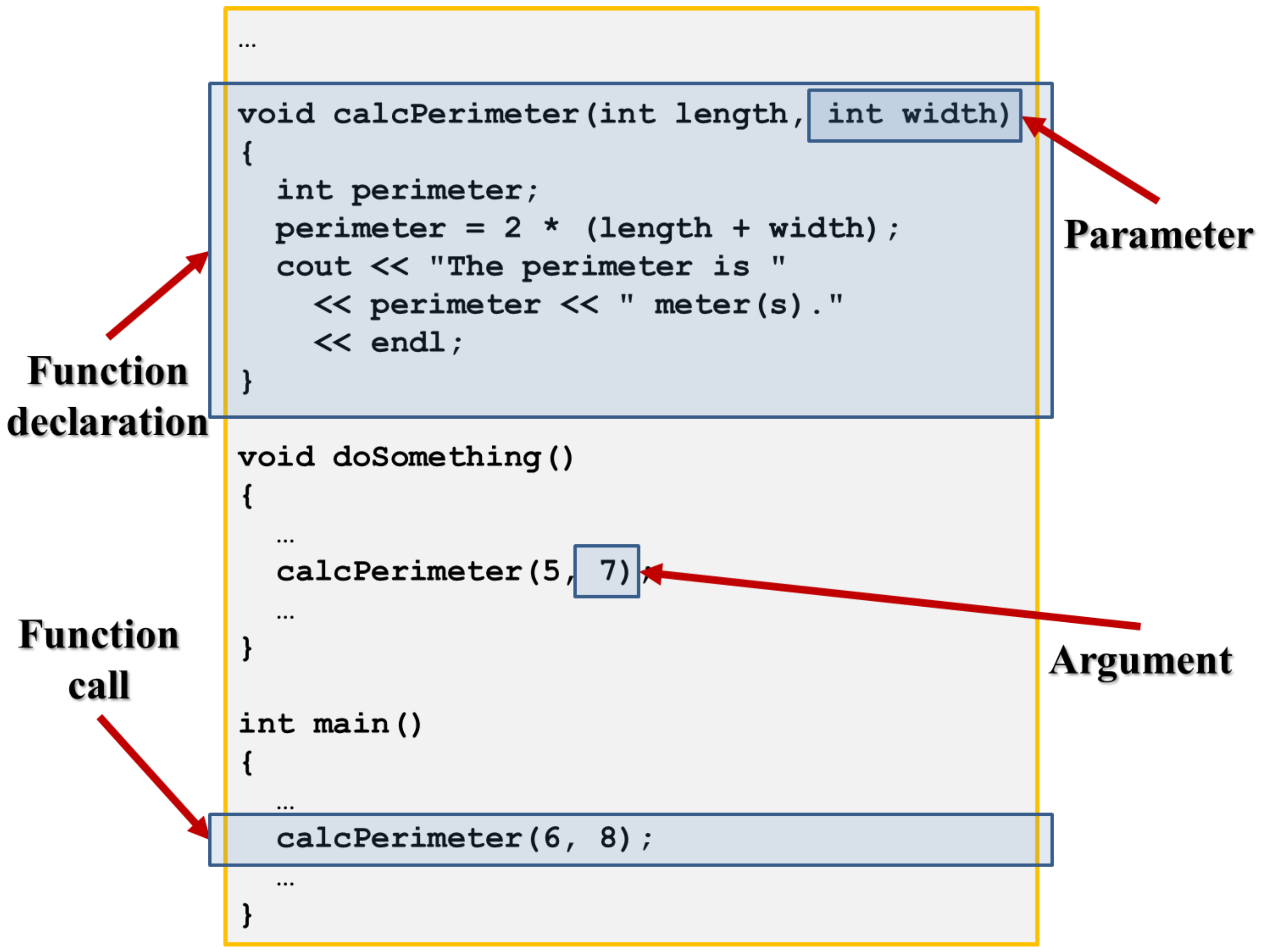
- Parameters are the inputs to a function.
- Parameters are defined at the top of a function.
- Each parameter is separated from the next one with a comma.
- A parameter list has syntax:

  <data-type> <parameter-1>, <data-type> <parameter-2>, ….
- A parameter is also known as a formal parameter.
- Just like function main, a programmer-defined function may declare its own variables. Since each function is separate from any other function, the same variable name may be used in multiple functions.

Arguments / Actual parameters

- Whenever a function is called, there must be one argument provided for each parameter.
- Parameters and arguments must match in number, order, and data type.
- An argument is also known as an actual parameter.

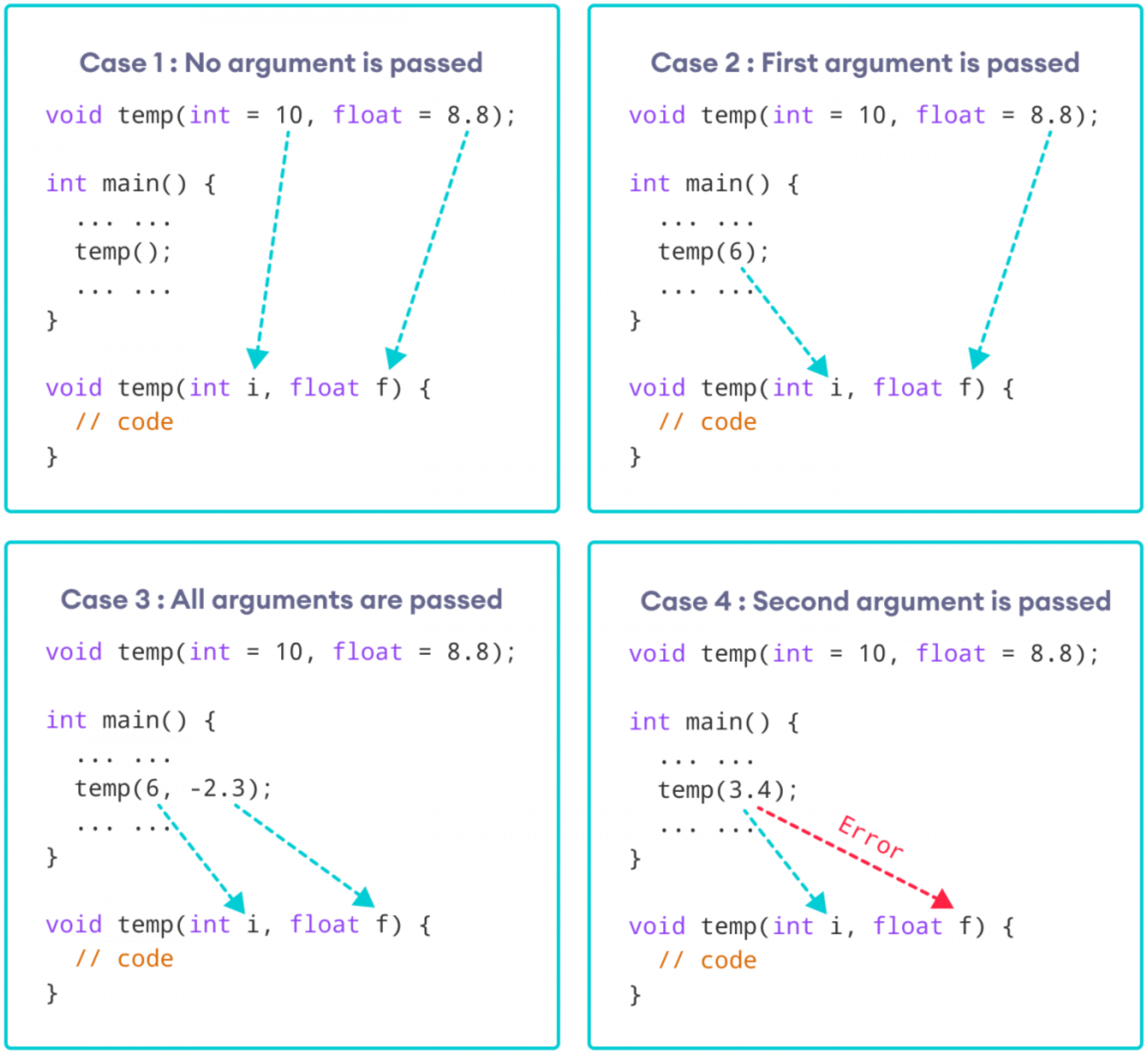
- Often times Arguments and Parameters are used interchangeably.



```
...

void calcPerimeter(int length, int width)
{
   int perimeter;
   perimeter = 2 * (length + width);
   cout << "The perimeter is "
     << perimeter << " meter(s)."
     << endl;
}

void doSomething()
{
   ...
   calcPerimeter(5, 7);
   ...
}

int main()
{
   ...
   calcPerimeter(6, 8);
   ...
}
```

Default Arguments (Parameters)

- If a function with default arguments is called without passing arguments, then the default parameters are used.
- However, if arguments are passed while calling the function, the default arguments are ignored.



Case 1 : No argument is passed

```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp();
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Case 2 : First argument is passed

```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp(6);
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Case 3 : All arguments are passed

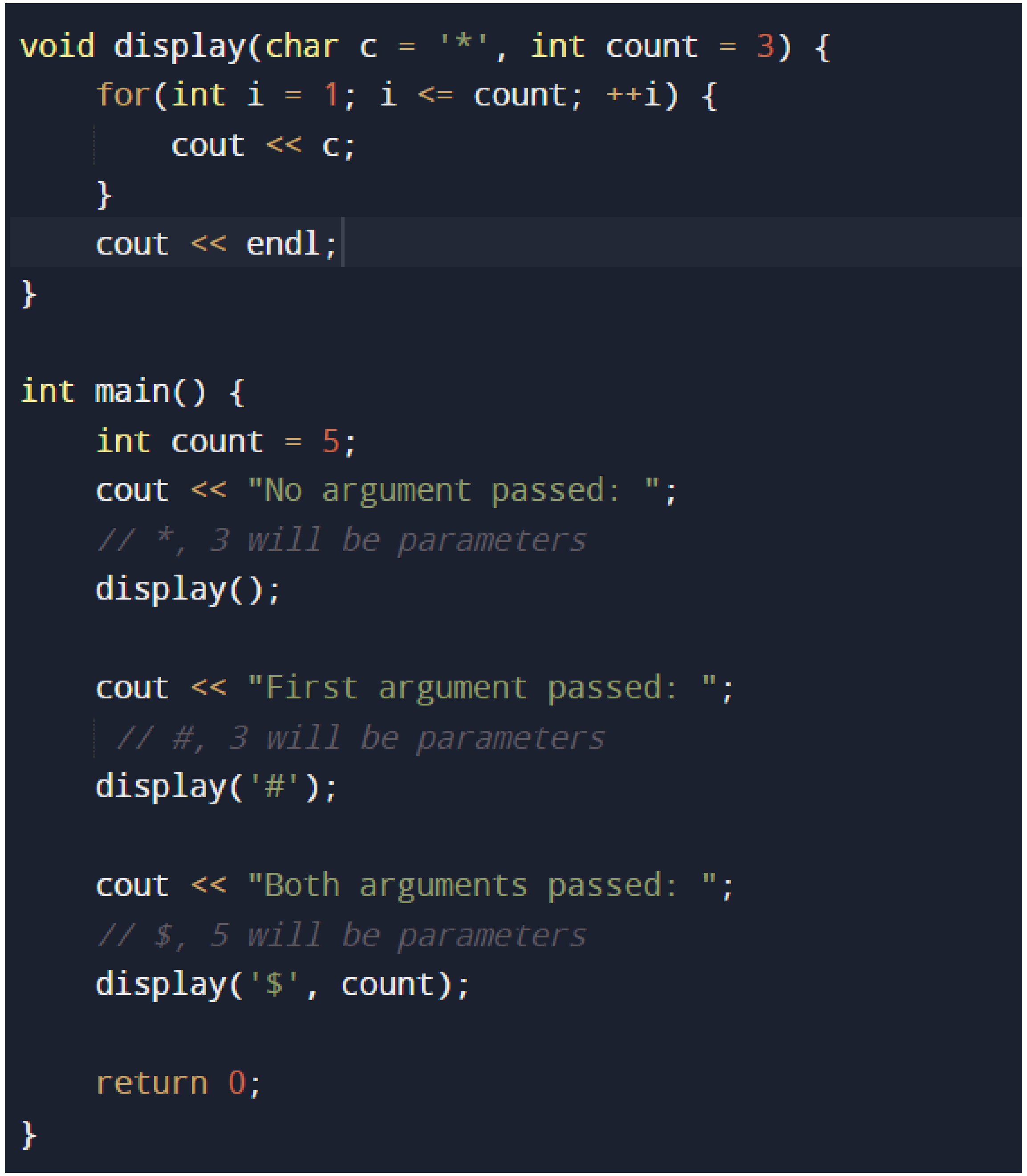
```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp(6, -2.3);
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Case 4 : Second argument is passed

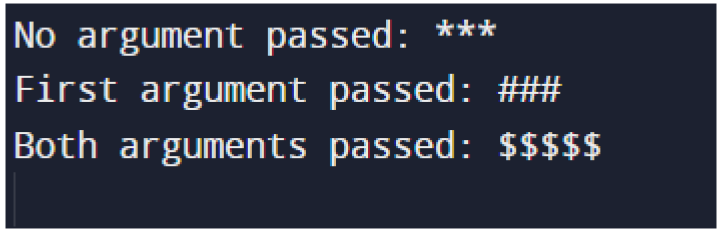
```
void temp(int = 10, float = 8.8);

int main() {
  ... ...
  temp(3.4);
  ... ...
}

void temp(int i, float f) {
  // code
}
```

Working of default arguments

1. When temp() is called, both the default parameters are used by the function.
2. When temp(6) is called, the first argument becomes 6 while the default value is used for the second parameter.
3. When temp(6, -2.3) is called, both the default parameters are overridden, resulting in i = 6 and f = -2.3.
4. When temp(3.4) is passed, the function behaves in an undesired way because the second argument cannot be passed without passing the first argument.



```cpp
void display(char c = '*', int count = 3) {
    for(int i = 1; i <= count; ++i) {
        cout << c;
    }
    cout << endl;
}

int main() {
    int count = 5;
    cout << "No argument passed: ";
    // *, 3 will be parameters
    display();

    cout << "First argument passed: ";
     // #, 3 will be parameters
    display('#');

    cout << "Both arguments passed: ";
    // $, 5 will be parameters
    display('$', count);

    return 0;
}
```



```
No argument passed: ***
First argument passed: ###
Both arguments passed: $$$$$
```