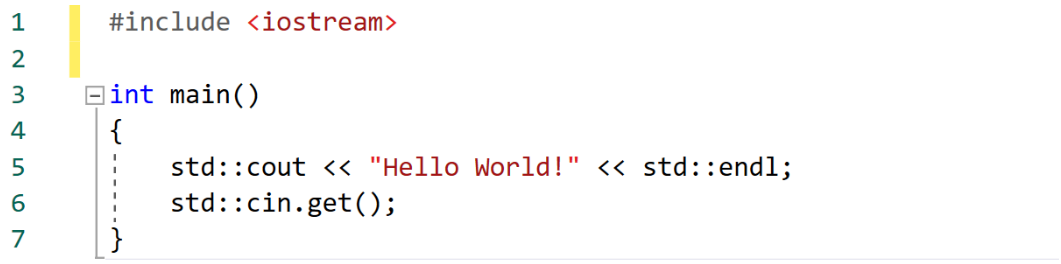
Hello world! in C++ by Chendy Lee

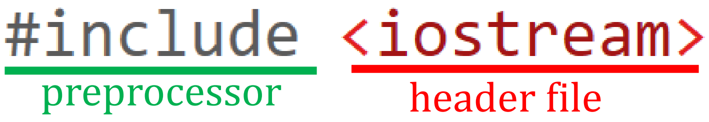


```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello World!" << std::endl;
6      std::cin.get();
7  }
```

row 1

#include is one of the preprocessor(前置處理器) directive(指示詞).Words who are added ahead with symbol “#” called “hash” are just preprocessors. Preprocessors’ function is “work” before the actual application of the complier. In this case, include is “to include something into this file” literally.

#include <iostream> means that it include or find a file called “iostream” and take all of the contents of that file pasting into the current programming file, and this file that included are typically called header file(標頭檔).

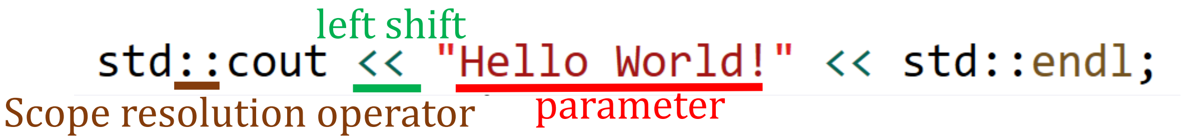


row 3

int main () is called the main function, which is the most important stuff for every C++ or C program. Main function is also called the entry point(入口點).Entry point is the point where starts first code execution when we run our application. As the program is running, the complier will execute the lines of the code we type in order.

In general, the integer function must return the value which the type is int(integer), but the main function is just an exception in this case. We don't need to return any kind of value from the main function, if we don't return anything, the main function will assume your return value is zero!

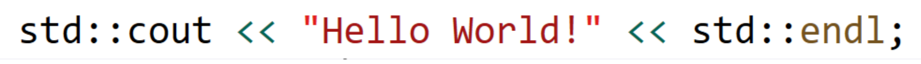
row5



```
std::cout << "Hello World!" << std::endl;
```

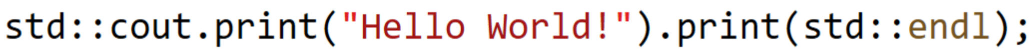
(i)<< is called "double left angle brackets" or "left shift", which is one of the operators. Every operator owns their unique function. But when we use some keywords to call or "declare" them, the operators will perform other functions different from their original functions, this kind of phenomenon is called "operator overloading"(運算子超載), and these kinds of operators are called "overloaded operators".

In this case, the keyword "**cout**" declares **<<**, so **<<** become a function, and there are parameters “Hello World” behind the operators. Therefore, we can think this long string which is



```
std::cout << "Hello World!" << std::endl;
```

as



```
std::cout.print("Hello World!").print(std::endl);
```

.

(Note: std::cout.print () don’t work in C++, it just explains the reasonable and acceptable function of std::cout <<.)

(ii)**cout**

Actually, **cout** means an output stream(輸出流) or the location where the word output. So, the operators << push the parameters “Hello World!” into **cout**, causing parameters to get printed to the console(控制台).

(iii)**endl**

The word **endl** is the abbreviation of **end line**. **endl** tells the console to advance to the next line(換行/進位).

(Note: std::endl also can replace with “\n”).

(iv)**std::cout**

There is a concept must be mentioned is called namespace. (命名空間). Namespace is the range of the signs could be identified.

Namespace is just like the name of a company, there are two employees, and share the same employee ID called “123”. But one is belonging to company A, and other is belonging to company B. Although they share the same ID, but they would not be mixed up. One is ID 123 in A, another is ID 123 in B. With namespace, the name of the function or identifier could not conflict. Since the program are used to keeping up with times, there are new identifiers (usually are the name of function) introduced into the standard library. But there are chances that the new identifiers may conflict with the original identifiers. In order to solve for this problem, C++ moves all the functionality in the standard library into a unified namespace called **std**. (short for standard.)

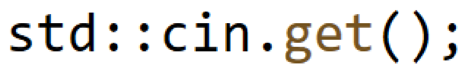
Output stream **cout** and input stream **cin** are the members of the namespace **std**. In order to construct the relationship between them and namespace. Then **::** must be introduced.

(v)**::**

Scope(範圍) is the context or association between value(值數) and expressions(表達式).The operator :: just derive from this kind of concept.

:: is called “ scope resolution operator”, its main function is to help complier to identify and specify the context or association between identifier and namespace. So, **std::cout** means we use cout this function **cout** which is taken from the namespace called **std**.

row6



```
std::cin.get();
```

(i)**cin**

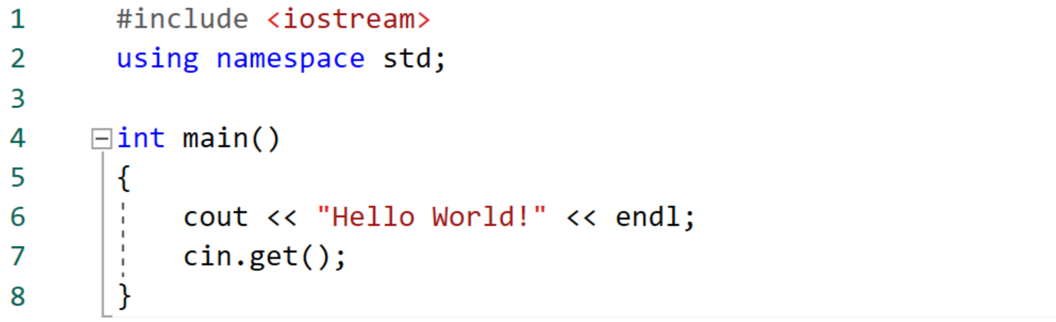
Compared to **cout** which is the function of output, **cin** is the function of input, so **cin** is also corresponding to the input stream. But since **cin** has finite function in input stream, it have some its own **built-in function**(內置函數) like cin.get(), cin.getline() to make up its disadvantage.

(ii)cin.get()

cin.get() is just function when we wait for us to press Enter key, and then the console would advance to the next line which is nothing which means the main function don’t return nothing which means it return zero which means the program executes successfully and finishes later.

(Note: If we write using namespace std; in advance, then we only write cout, endl, cin instead of std::cout, std::endl, std::cin .

But it had better not be used! It is highly discouraged!)



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

int main()
{
    cout << "Hello World!" << endl;
    cin.get();
}
```

using directive is used to tell the complier to check the mentioned namespace when trying to solve any identifier having no namespace prefix.

In this case, **cout** has no namespace in this console, so the compliers would consider **cout** should be **std::cout**, which means **cout** in the namespace **std**.