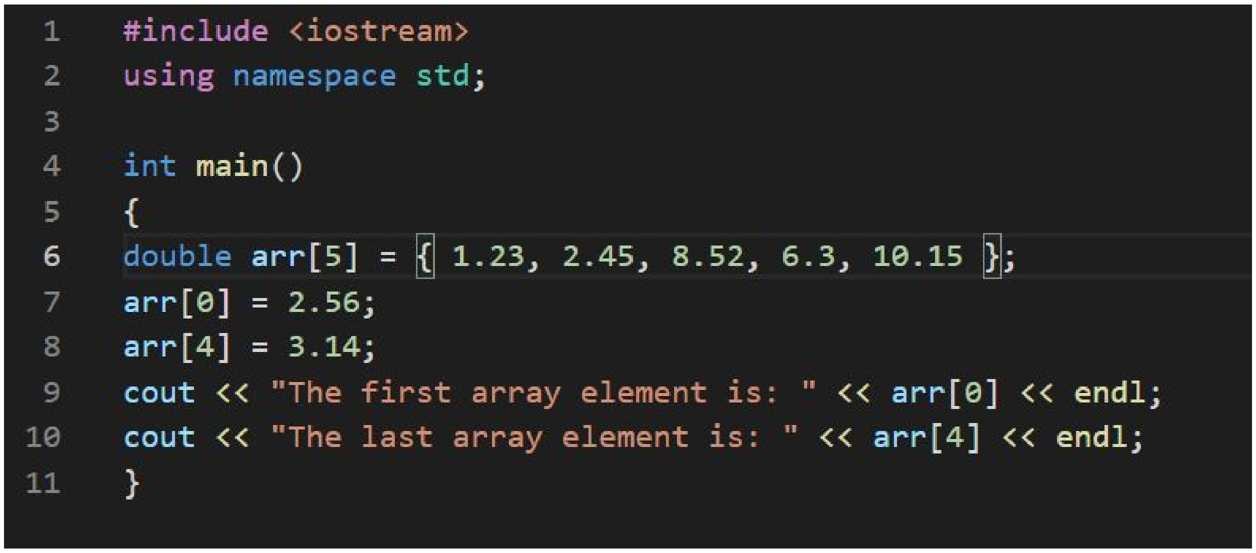
Name: Nguyen Anh Tai
Student ID: 16128
Task: Reading Assignment

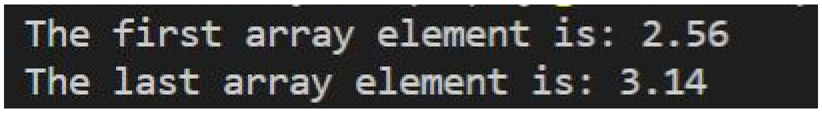
CHAPTER 15: EXERCISES

15.1 ARRAY DEFINITION



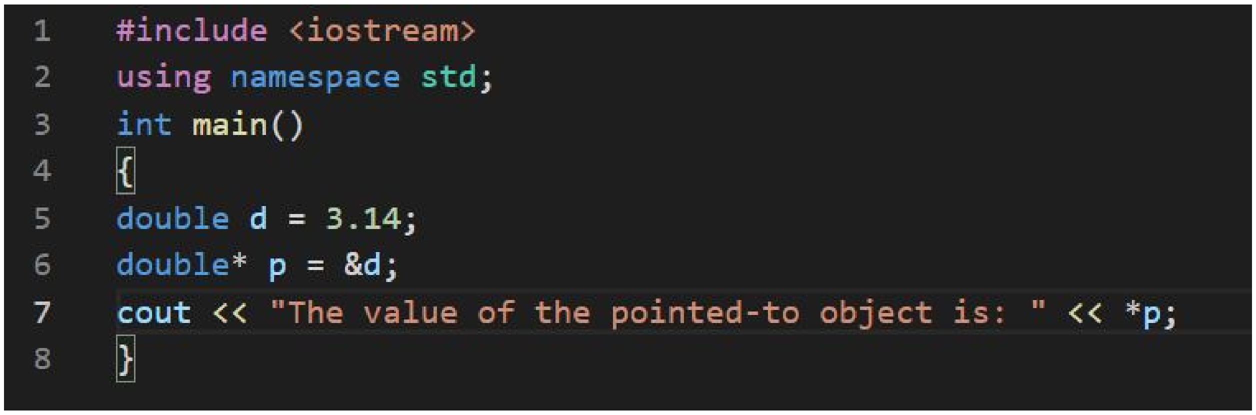
```cpp
#include <iostream>
using namespace std;

int main()
{
double arr[5] = { 1.23, 2.45, 8.52, 6.3, 10.15 };
arr[0] = 2.56;
arr[4] = 3.14;
cout << "The first array element is: " << arr[0] << endl;
cout << "The last array element is: " << arr[4] << endl;
}
```

Output:



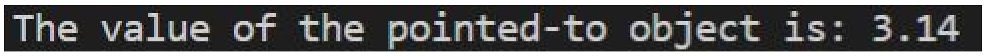
```
The first array element is: 2.56
The last array element is: 3.14
```

15.2 POINTER TO AN OBJECT



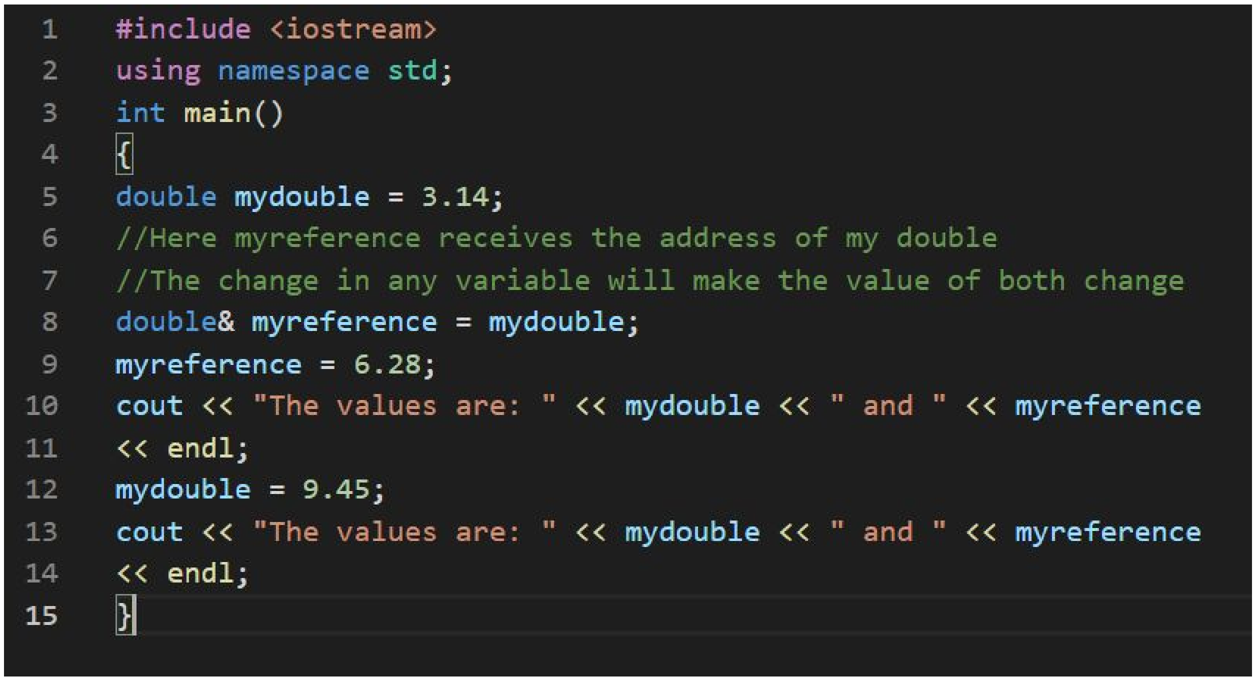
```cpp
#include <iostream>
using namespace std;
int main()
{
double d = 3.14;
double* p = &d;
cout << "The value of the pointed-to object is: " << *p;
}
```

Output:



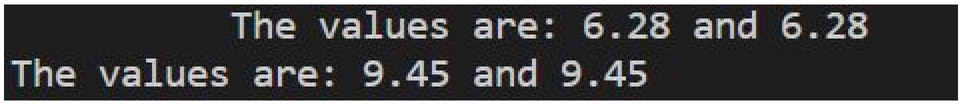
```
The value of the pointed-to object is: 3.14
```

15.3 REFERENCE TYPE



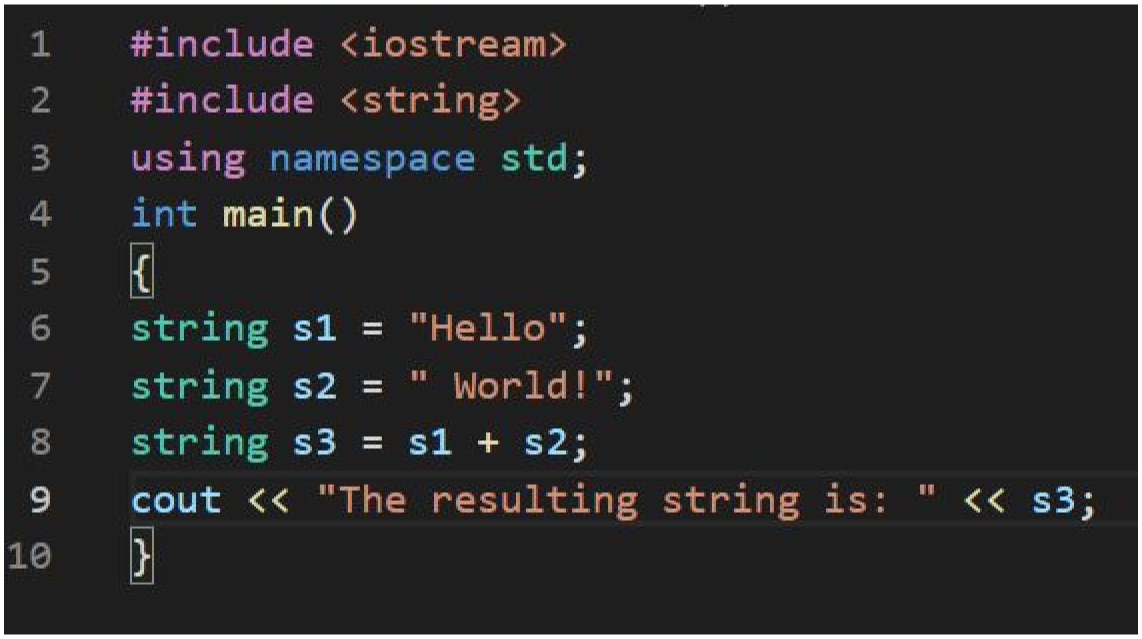
```cpp
#include <iostream>
using namespace std;
int main()
{
double mydouble = 3.14;
//Here myreference receives the address of my double
//The change in any variable will make the value of both change
double& myreference = mydouble;
myreference = 6.28;
cout << "The values are: " << mydouble << " and " << myreference
<< endl;
mydouble = 9.45;
cout << "The values are: " << mydouble << " and " << myreference
<< endl;
}
```

Output:



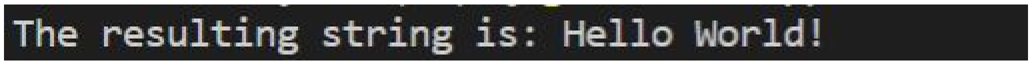
```
          The values are: 6.28 and 6.28
The values are: 9.45 and 9.45
```

15.4 STRINGS



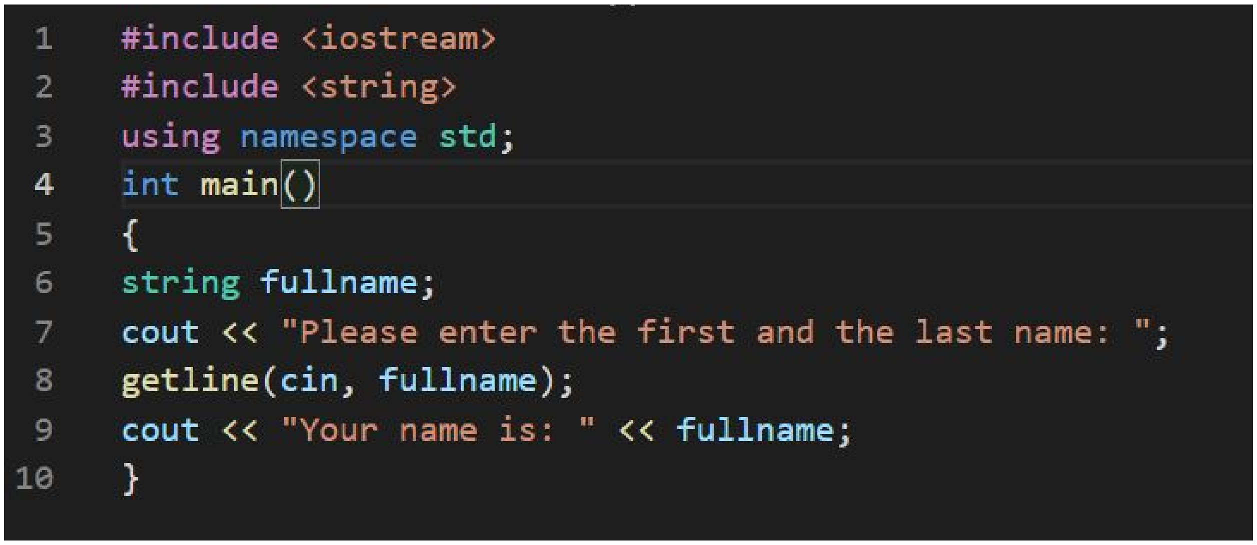
```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
string s1 = "Hello";
string s2 = " World!";
string s3 = s1 + s2;
cout << "The resulting string is: " << s3;
}
```

Output:

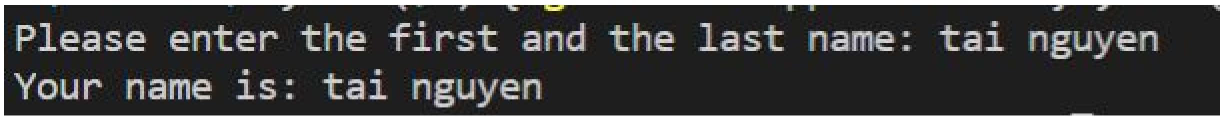


```
The resulting string is: Hello World!
```

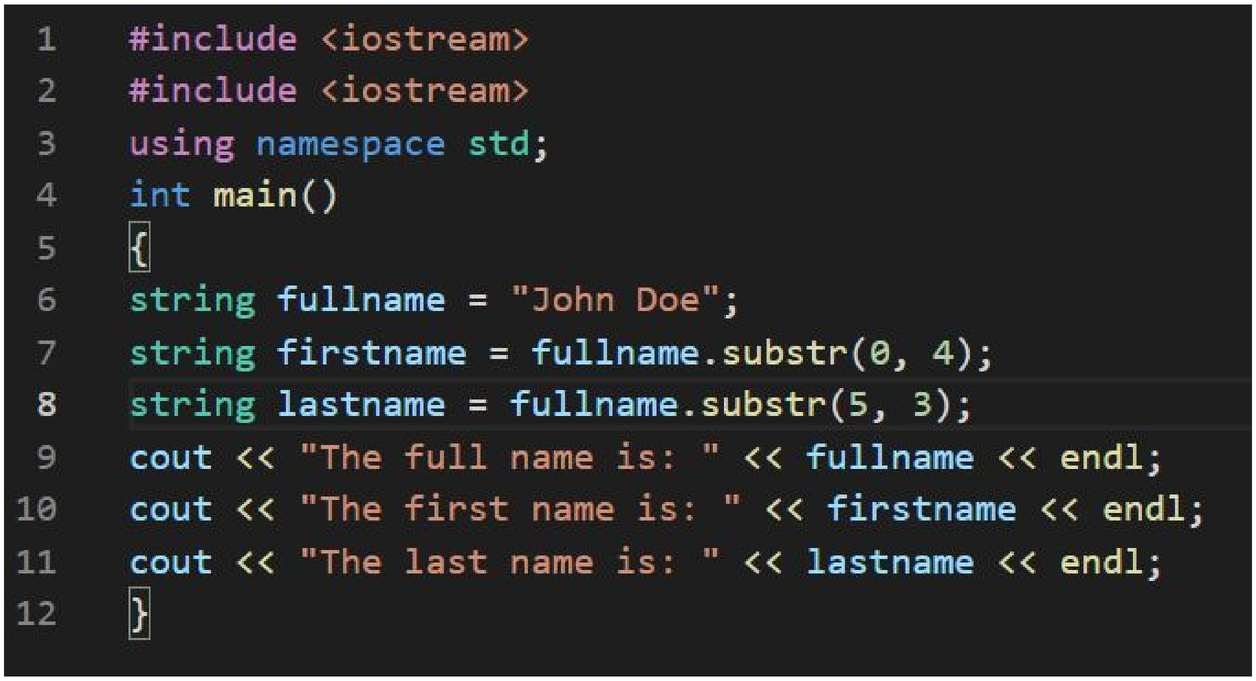
15.5 STRINGS FROM STANDARD INPUT


```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{
string fullname;
cout << "Please enter the first and the last name: ";
getline(cin, fullname);
cout << "Your name is: " << fullname;
}
```

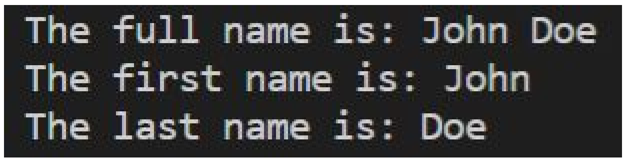
Output:


```
Please enter the first and the last name: tai nguyen
Your name is: tai nguyen
```

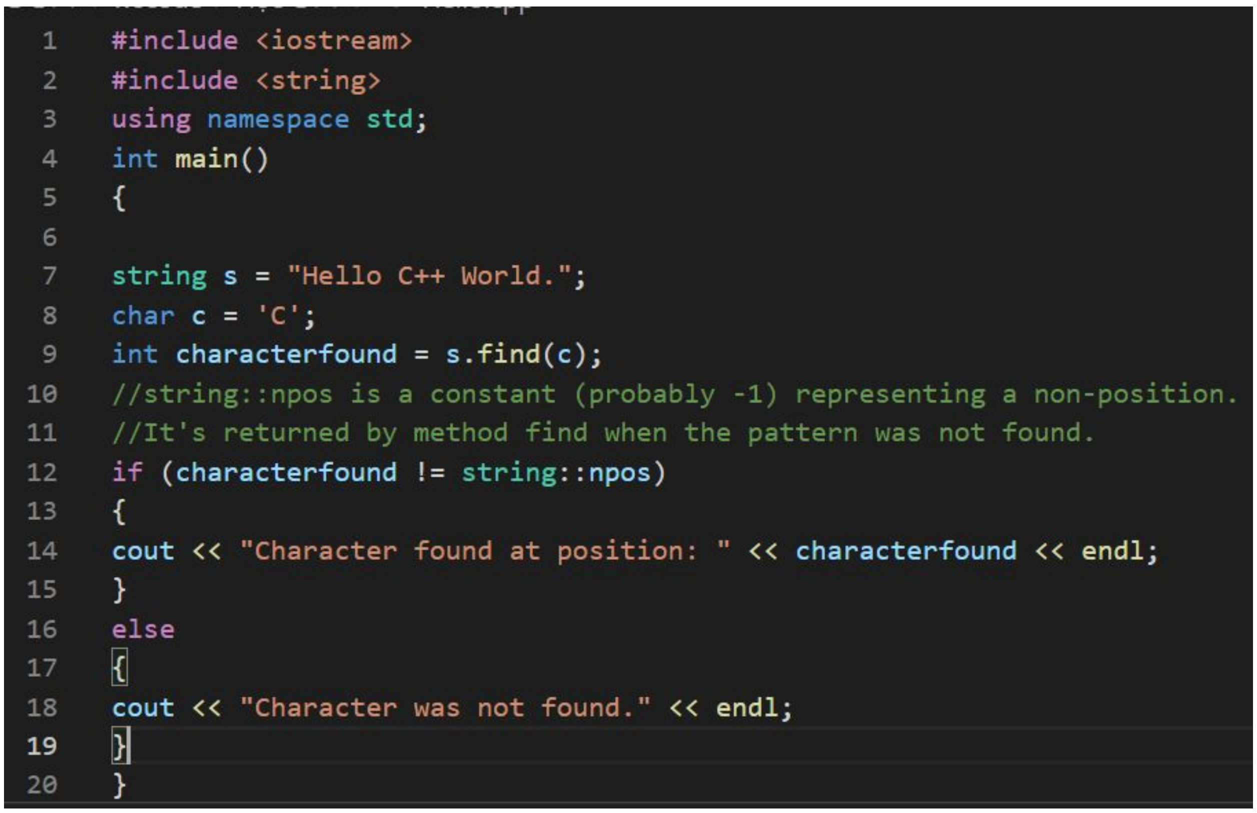
15.6 CREATING A SUBSTRING


```cpp
#include <iostream>
#include <iostream>
using namespace std;
int main()
{
string fullname = "John Doe";
string firstname = fullname.substr(0, 4);
string lastname = fullname.substr(5, 3);
cout << "The full name is: " << fullname << endl;
cout << "The first name is: " << firstname << endl;
cout << "The last name is: " << lastname << endl;
}
```

Output:

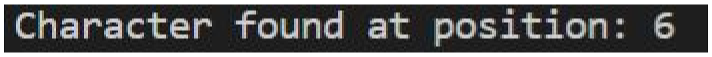

```
The full name is: John Doe
The first name is: John
The last name is: Doe
```

15.7 FINDING A SINGLE CHARACTER



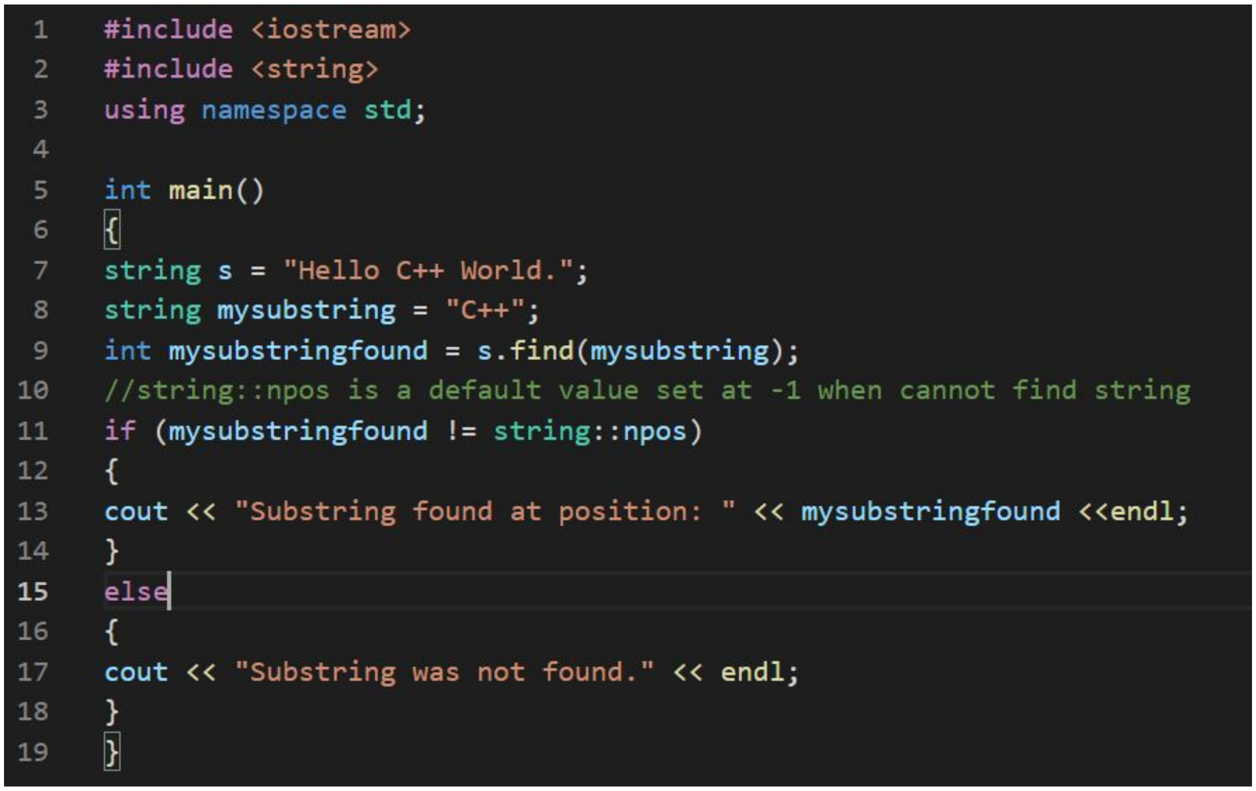
```cpp
#include <iostream>
#include <string>
using namespace std;
int main()
{

string s = "Hello C++ World.";
char c = 'C';
int characterfound = s.find(c);
//string::npos is a constant (probably -1) representing a non-position.
//It's returned by method find when the pattern was not found.
if (characterfound != string::npos)
{
cout << "Character found at position: " << characterfound << endl;
}
else
{
cout << "Character was not found." << endl;
}
}
```

Output:



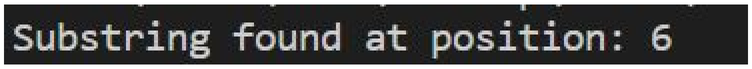
```
Character found at position: 6
```

15.8 FINDING A SUBSTRING



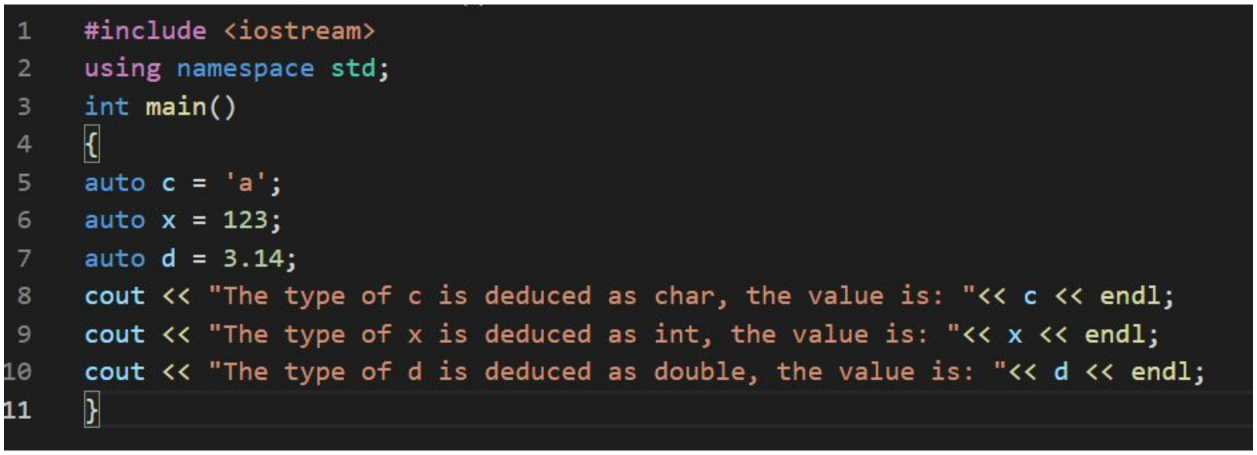
```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
string s = "Hello C++ World.";
string mysubstring = "C++";
int mysubstringfound = s.find(mysubstring);
//string::npos is a default value set at -1 when cannot find string
if (mysubstringfound != string::npos)
{
cout << "Substring found at position: " << mysubstringfound <<endl;
}
else
{
cout << "Substring was not found." << endl;
}
}
```

Output:



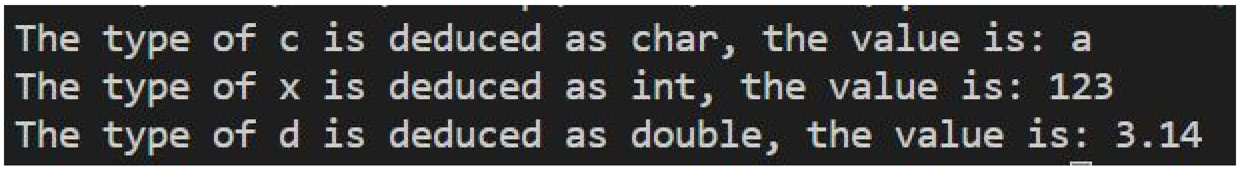
```
Substring found at position: 6
```

15.9 AUTOMATIC TYPE DEDUCTION



```cpp
#include <iostream>
using namespace std;
int main()
{
auto c = 'a';
auto x = 123;
auto d = 3.14;
cout << "The type of c is deduced as char, the value is: "<< c << endl;
cout << "The type of x is deduced as int, the value is: "<< x << endl;
cout << "The type of d is deduced as double, the value is: "<< d << endl;
}
```

Output:



```
The type of c is deduced as char, the value is: a
The type of x is deduced as int, the value is: 123
The type of d is deduced as double, the value is: 3.14
```

CHAPTER 16: STATEMENT

16.1 SELECTION STATEMENTS

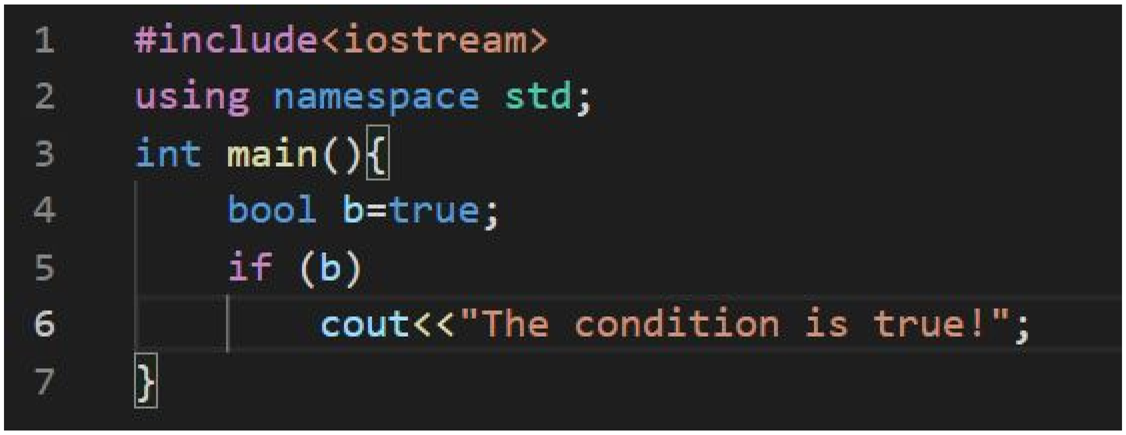
16.1.1 IF STATEMENT

- The format of if statement is below here:

```
if (condition) statement
```

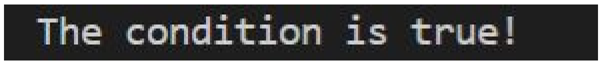
The statement executes only if the condition is true. Example:

Example:



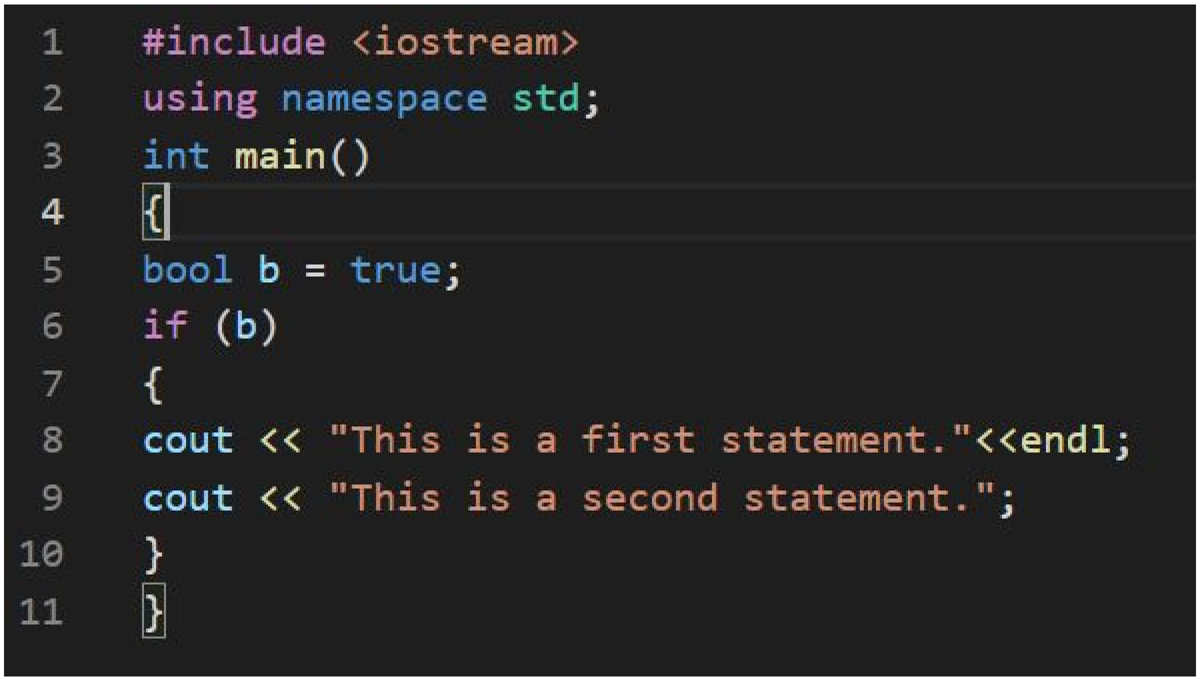
```cpp
#include<iostream>
using namespace std;
int main(){
    bool b=true;
    if (b)
        cout<<"The condition is true!";
}
```

Output:



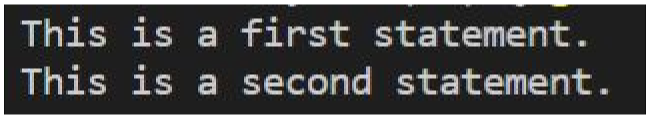
```
The condition is true!
```

Example:



```cpp
#include <iostream>
using namespace std;
int main()
{
bool b = true;
if (b)
{
cout << "This is a first statement."<<endl;
cout << "This is a second statement.";
}
}
```

Output:



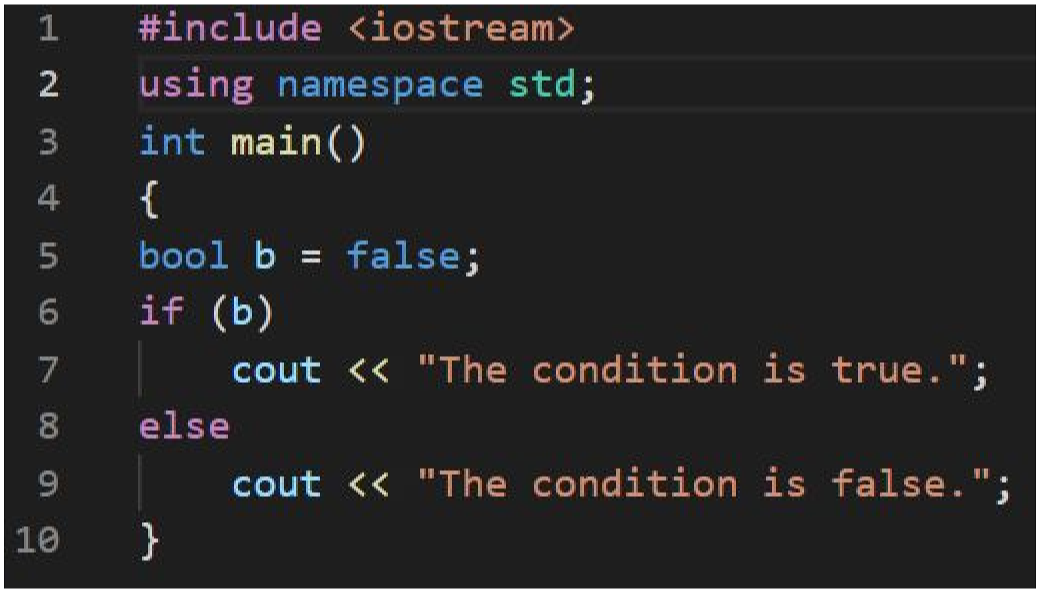
```
This is a first statement.
This is a second statement.
```

- Another form is if-else statement
  - Format:

```
if (condition) statement else statement
```

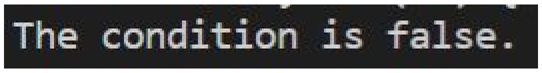
If the condition is true, the first statement executes, otherwise the second statement after the else keyword executes. Example:

- Example:



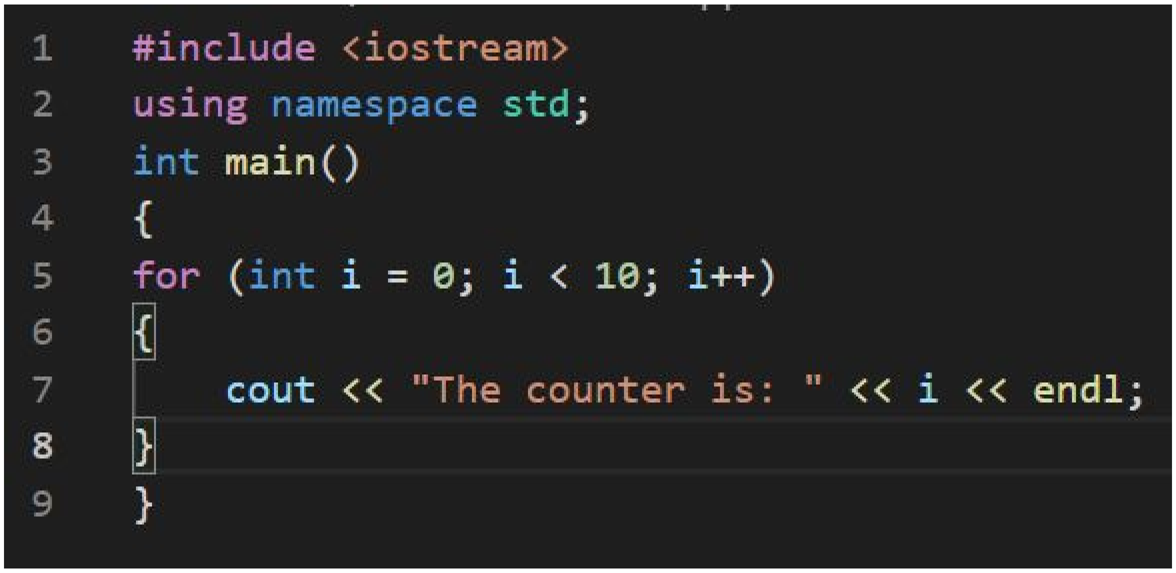
```cpp
#include <iostream>
using namespace std;
int main()
{
bool b = false;
if (b)
    cout << "The condition is true.";
else
    cout << "The condition is false.";
}
```

Output:



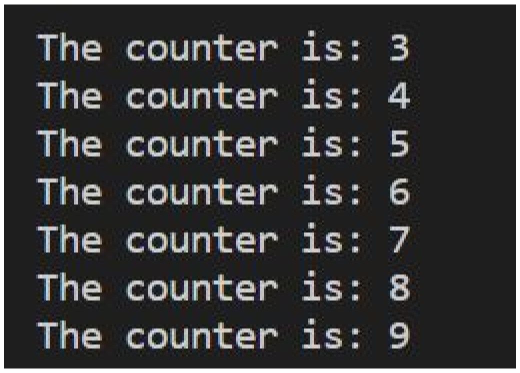
```
The condition is false.
```

16.2.1 FOR STATEMENT



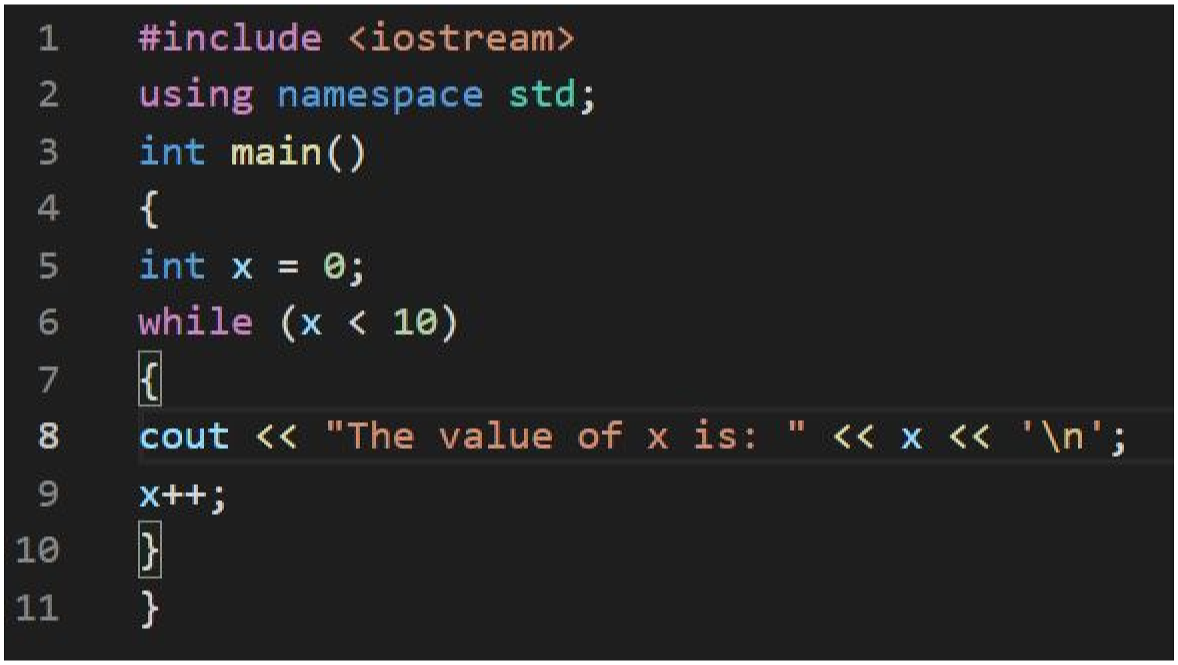
```cpp
#include <iostream>
using namespace std;
int main()
{
for (int i = 0; i < 10; i++)
{
    cout << "The counter is: " << i << endl;
}
}
```

Output:



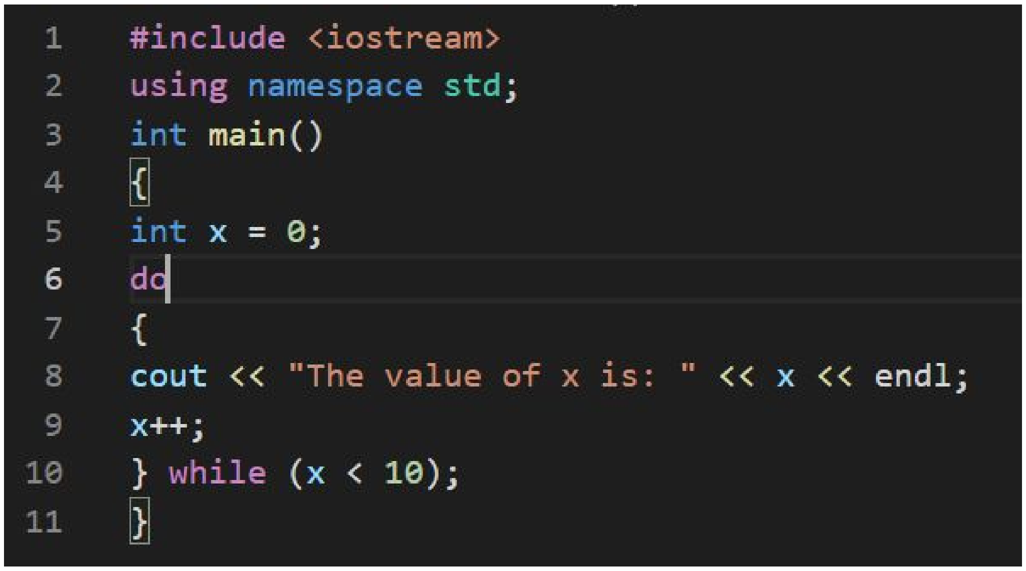
```
The counter is: 3
The counter is: 4
The counter is: 5
The counter is: 6
The counter is: 7
The counter is: 8
The counter is: 9
```

16.2.2 WHILE STATEMENT



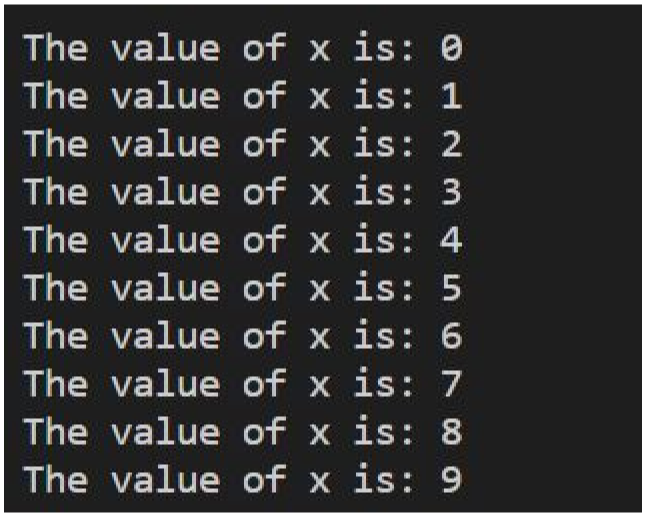
```cpp
#include <iostream>
using namespace std;
int main()
{
int x = 0;
while (x < 10)
{
cout << "The value of x is: " << x << '\n';
x++;
}
}
```

16.2.3 DO STATEMENT



```cpp
#include <iostream>
using namespace std;
int main()
{
int x = 0;
do
{
cout << "The value of x is: " << x << endl;
x++;
} while (x < 10);
}
```

Output:



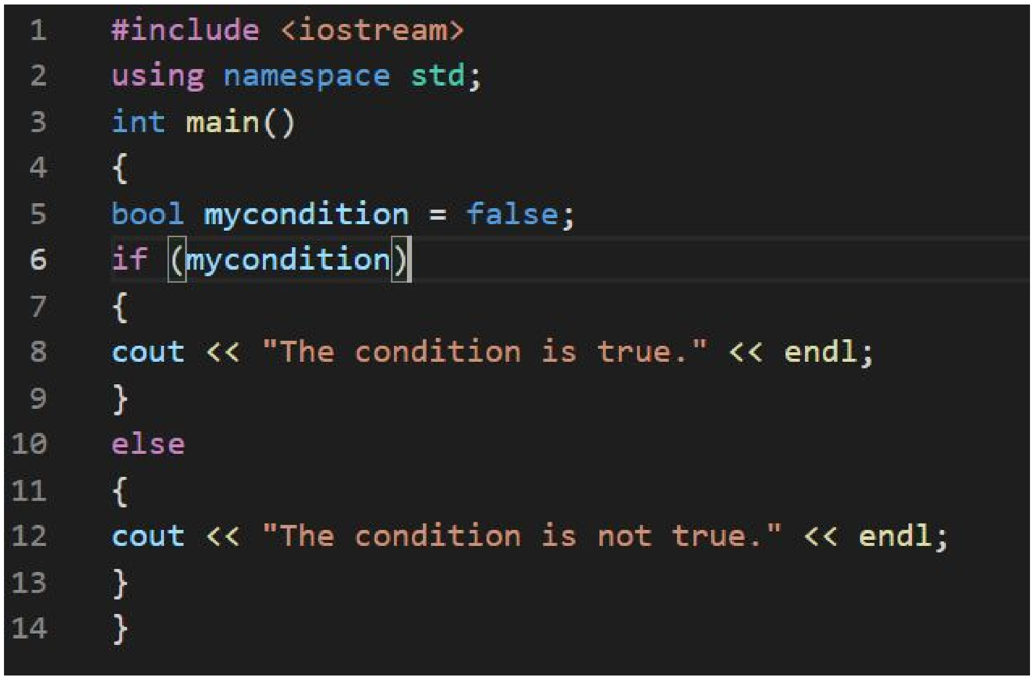
```
The value of x is: 0
The value of x is: 1
The value of x is: 2
The value of x is: 3
The value of x is: 4
The value of x is: 5
The value of x is: 6
The value of x is: 7
The value of x is: 8
The value of x is: 9
```

CHAPTER 17: CONSTANT

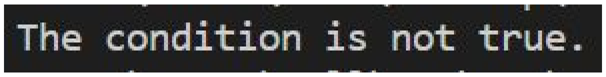
- When we want to have a read-only object or promise not to change the value of some object in the current scope, we make it a constant. C++ uses the const type qualifier to mark the object as a read-only.
- Constants are not modifiable, attempt to do so results in a compile-time error

CHAPTER 18: EXERCISES

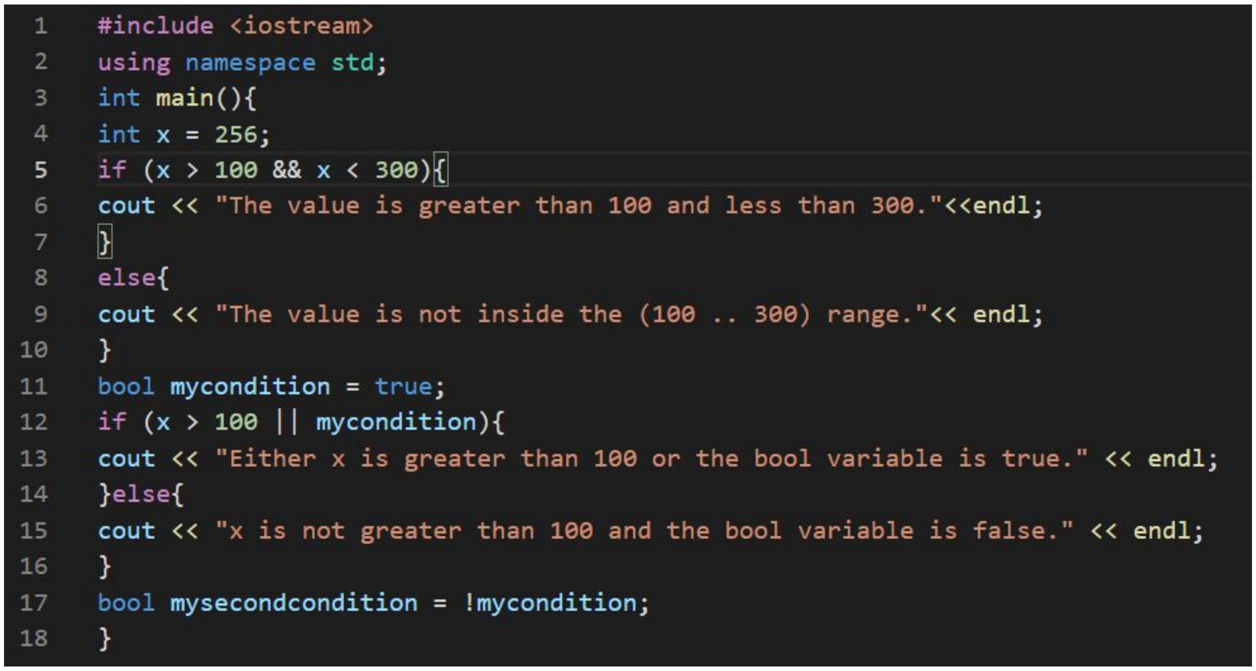
18.1 A SIMPLE IF STATEMENT


```cpp
#include <iostream>
using namespace std;
int main()
{
bool mycondition = false;
if (mycondition)
{
cout << "The condition is true." << endl;
}
else
{
cout << "The condition is not true." << endl;
}
}
```

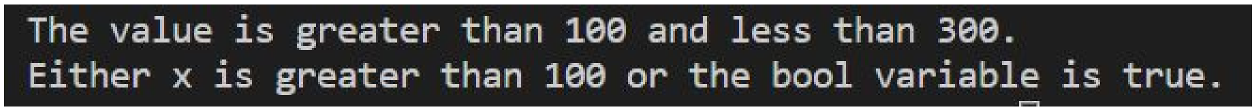
Output:


```
The condition is not true.
```

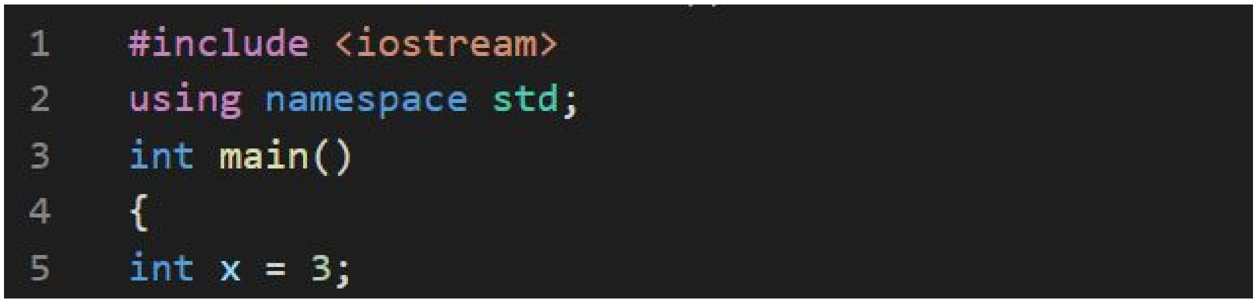
18.2 LOGICAL OPERATOR


```cpp
#include <iostream>
using namespace std;
int main(){
int x = 256;
if (x > 100 && x < 300){
cout << "The value is greater than 100 and less than 300."<<endl;
}
else{
cout << "The value is not inside the (100 .. 300) range."<< endl;
}
bool mycondition = true;
if (x > 100 || mycondition){
cout << "Either x is greater than 100 or the bool variable is true." << endl;
}else{
cout << "x is not greater than 100 and the bool variable is false." << endl;
}
bool mysecondcondition = !mycondition;
}
```

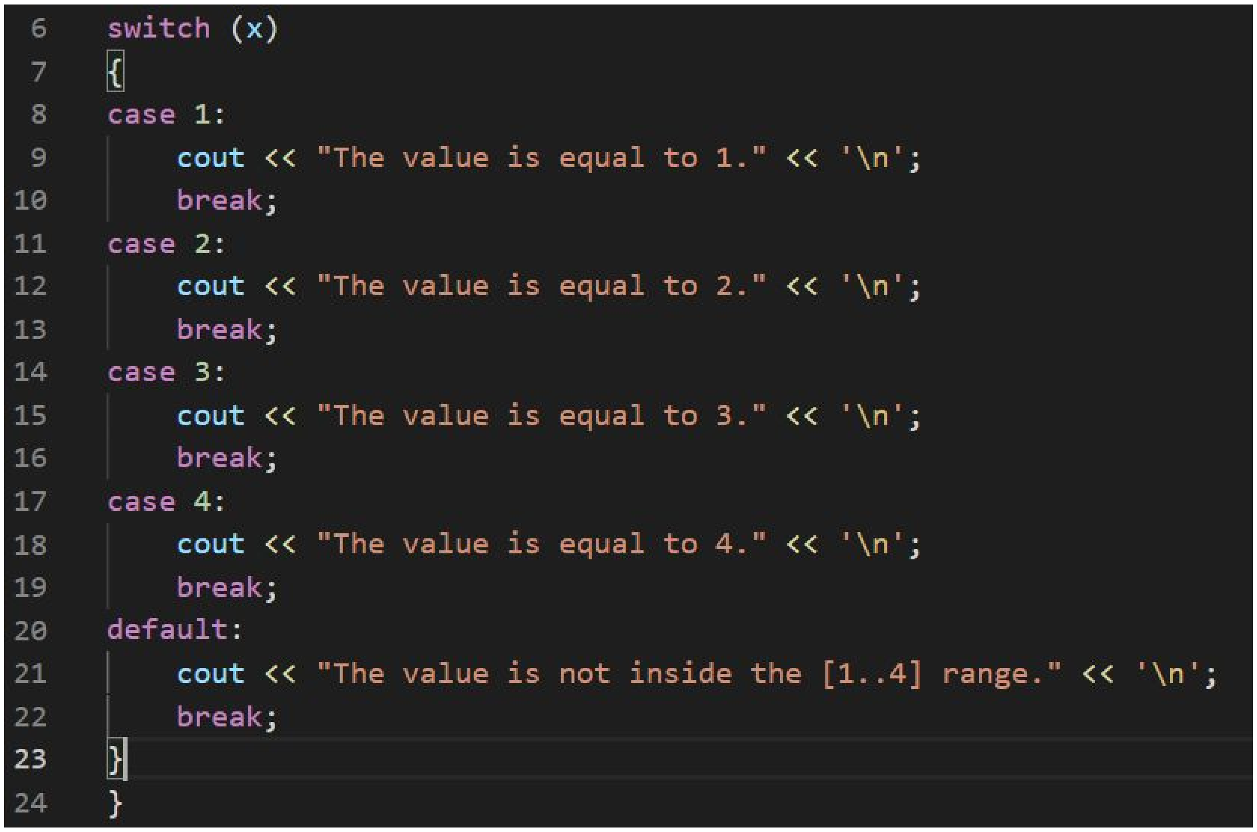
Output:


```
The value is greater than 100 and less than 300.
Either x is greater than 100 or the bool variable is true.
```

18.3 THE SWITCH STATEMENT

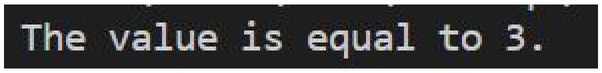


```cpp
#include <iostream>
using namespace std;
int main()
{
int x = 3;
```



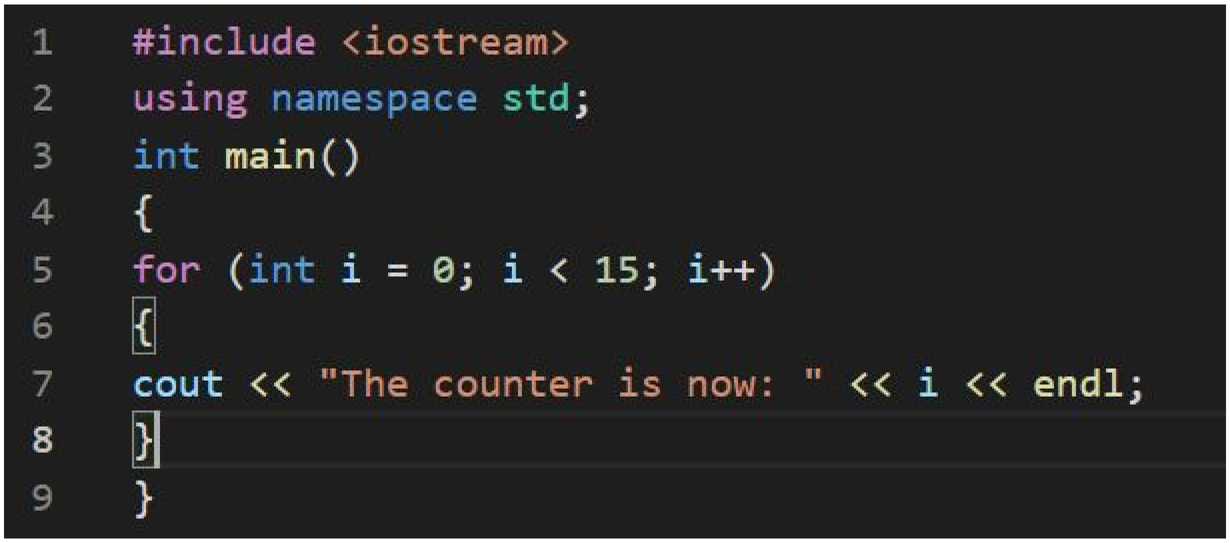
```cpp
switch (x)
{
case 1:
    cout << "The value is equal to 1." << '\n';
    break;
case 2:
    cout << "The value is equal to 2." << '\n';
    break;
case 3:
    cout << "The value is equal to 3." << '\n';
    break;
case 4:
    cout << "The value is equal to 4." << '\n';
    break;
default:
    cout << "The value is not inside the [1..4] range." << '\n';
    break;
}
}
```

Output:



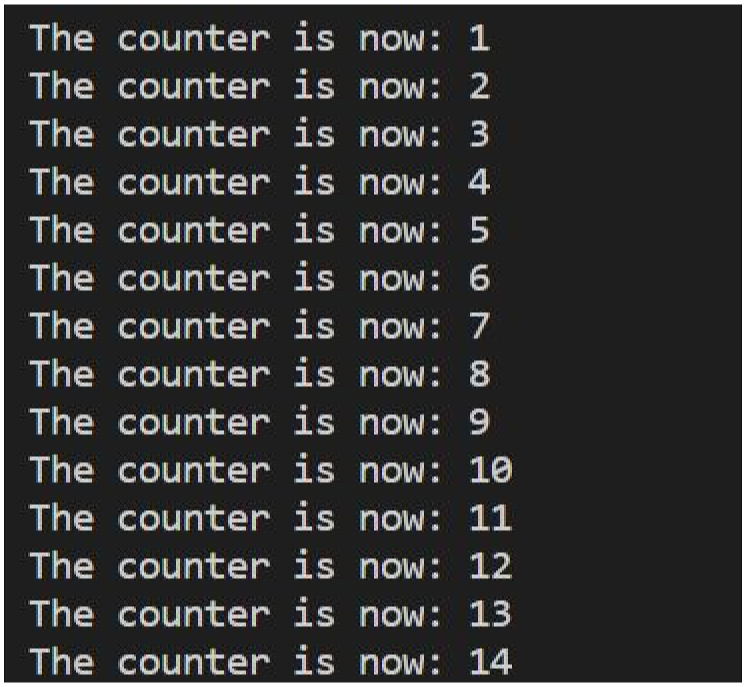
```
The value is equal to 3.
```

18.4 THE FOR LOOP



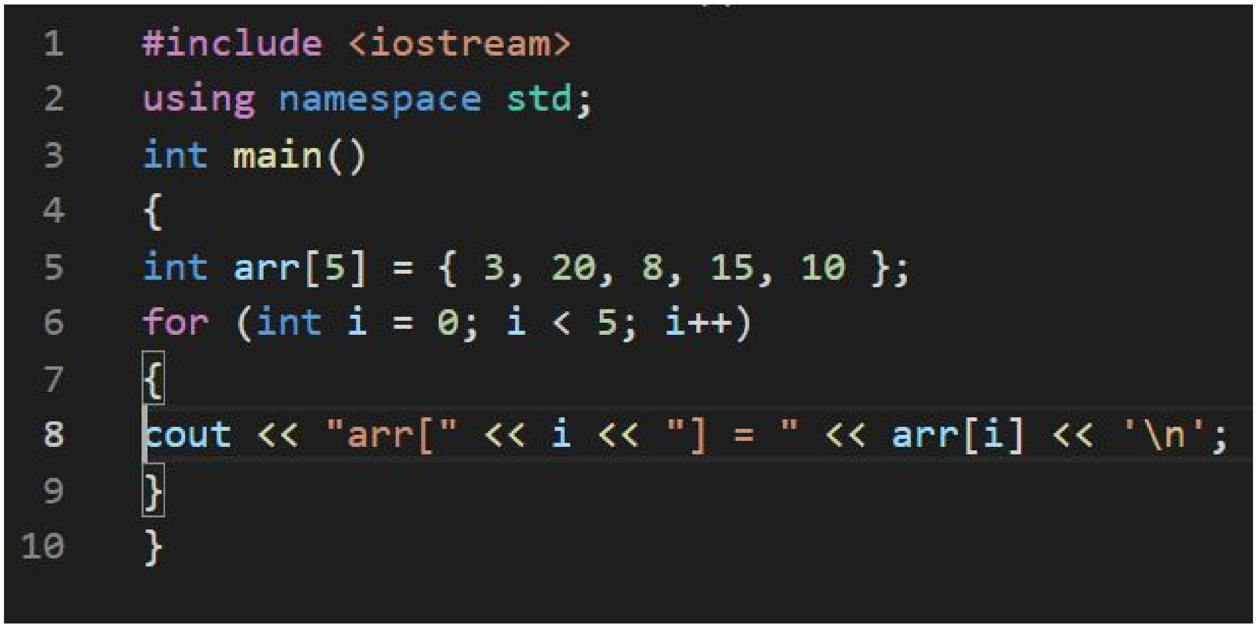
```cpp
#include <iostream>
using namespace std;
int main()
{
for (int i = 0; i < 15; i++)
{
cout << "The counter is now: " << i << endl;
}
}
```

Output:



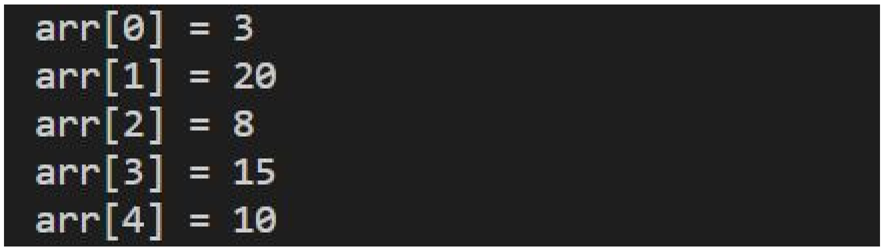
```
The counter is now: 1
The counter is now: 2
The counter is now: 3
The counter is now: 4
The counter is now: 5
The counter is now: 6
The counter is now: 7
The counter is now: 8
The counter is now: 9
The counter is now: 10
The counter is now: 11
The counter is now: 12
The counter is now: 13
The counter is now: 14
```

18.5 ARRAY AND THE FOR LOOP



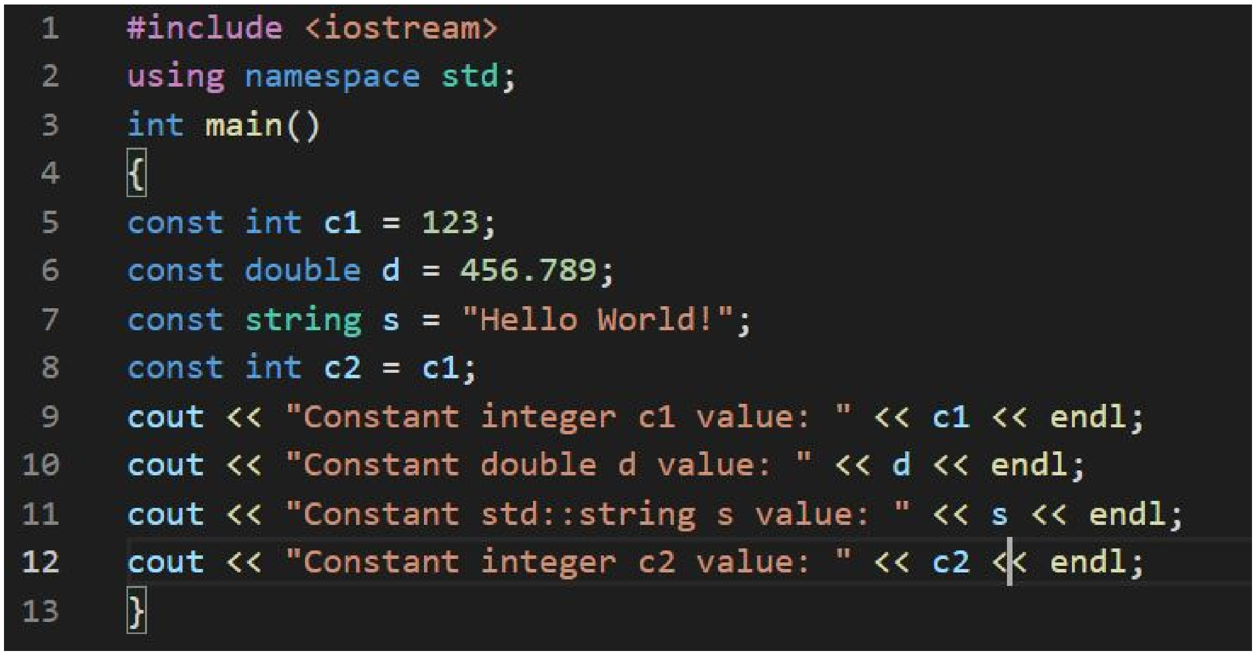
```cpp
#include <iostream>
using namespace std;
int main()
{
int arr[5] = { 3, 20, 8, 15, 10 };
for (int i = 0; i < 5; i++)
{
cout << "arr[" << i << "] = " << arr[i] << '\n';
}
}
```

Output:



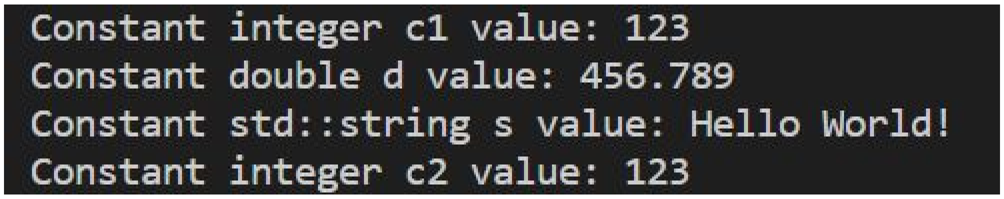
```
arr[0] = 3
arr[1] = 20
arr[2] = 8
arr[3] = 15
arr[4] = 10
```

18.6 THE CONST TYPE QUALIFIER



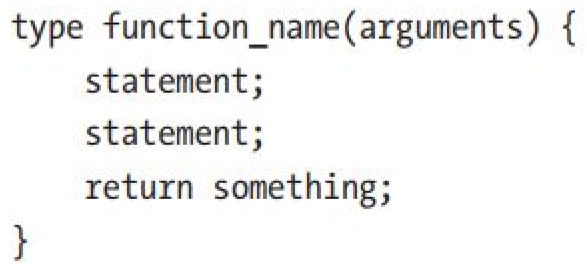
```cpp
#include <iostream>
using namespace std;
int main()
{
const int c1 = 123;
const double d = 456.789;
const string s = "Hello World!";
const int c2 = c1;
cout << "Constant integer c1 value: " << c1 << endl;
cout << "Constant double d value: " << d << endl;
cout << "Constant std::string s value: " << s << endl;
cout << "Constant integer c2 value: " << c2 << endl;
}
```

Output:


```
Constant integer c1 value: 123
Constant double d value: 456.789
Constant std::string s value: Hello World!
Constant integer c2 value: 123
```

CHAPTER 19: FUNCTIONS

19.1 INTRODUCTION

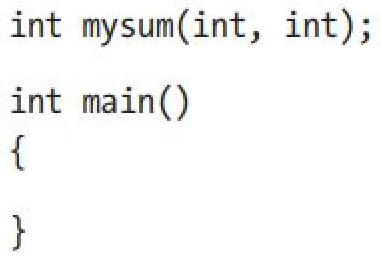

```
type function_name(arguments) {
    statement;
    statement;
    return something;
}
```

19.2 FUNCTION DECLARATION

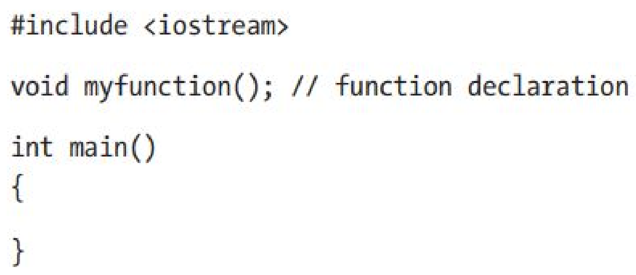
- To declare a function, we need to specify a return type, a name, and a list of parameters, if any. To declare a function called myfunction of type void that accepts no parameters. Specifically,

```
void myvoidfunction();
int main()
{
}
```

- Type void is a type that represents nothing, an empty set of values.
- Type int is a type that returns an integer value.
- In function declaration only, we can omit the parameters names, but we need to specify their types:

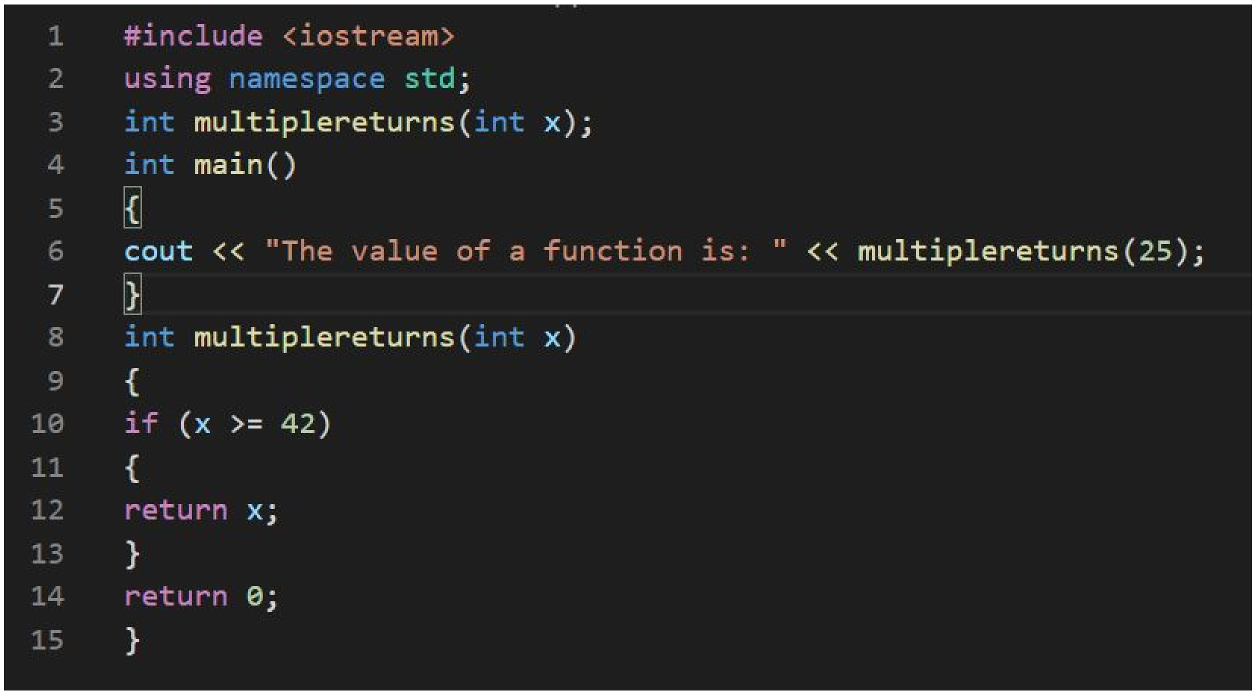

```
int mysum(int, int);

int main()
{

}
```

19.3 FUNCTION DEFINITION



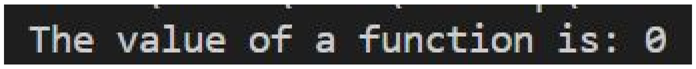
```cpp
#include <iostream>

void myfunction(); // function declaration

int main()
{

}
```

19.4 RETURN STATEMENT



```cpp
#include <iostream>
using namespace std;
int multiplereturns(int x);
int main()
{
cout << "The value of a function is: " << multiplereturns(25);
}
int multiplereturns(int x)
{
if (x >= 42)
{
return x;
}
return 0;
}
```

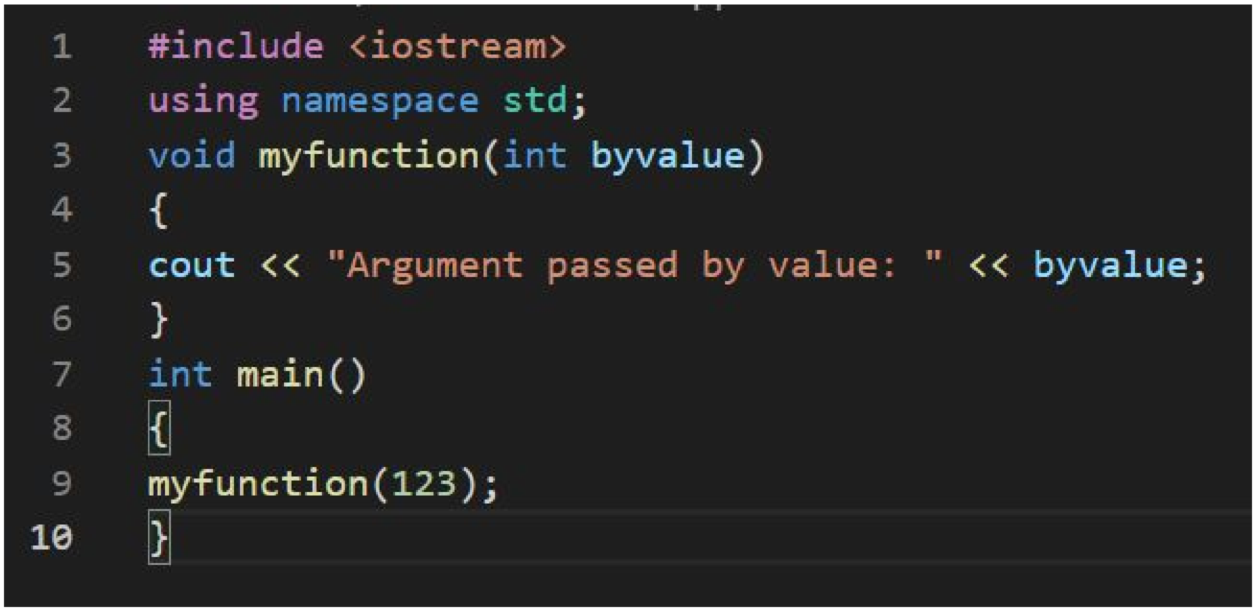
Output:



```
The value of a function is: 0
```

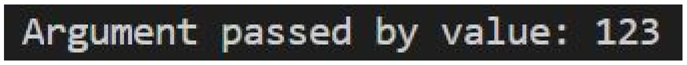
19.5.1 PASSING BY VALUE/COPY

Example:



```cpp
#include <iostream>
using namespace std;
void myfunction(int byvalue)
{
cout << "Argument passed by value: " << byvalue;
}
int main()
{
myfunction(123);
}
```

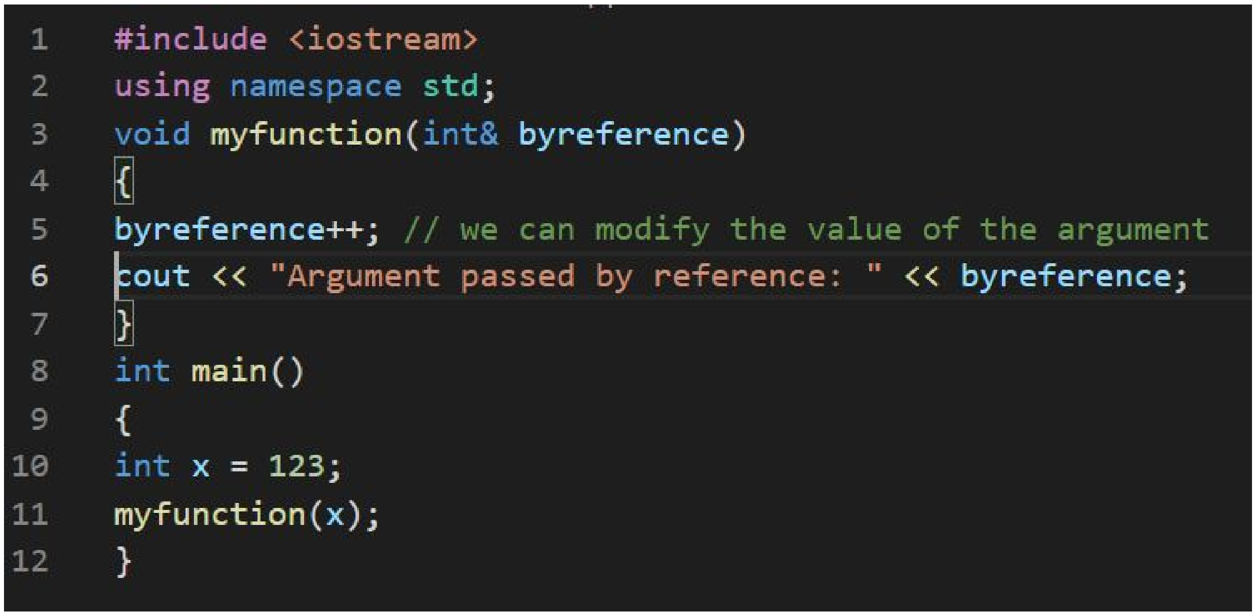
Output:



```
Argument passed by value: 123
```

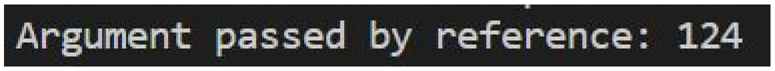
19.5.2 PASSING BY REFERENCE

Example:



```cpp
#include <iostream>
using namespace std;
void myfunction(int& byreference)
{
byreference++; // we can modify the value of the argument
cout << "Argument passed by reference: " << byreference;
}
int main()
{
int x = 123;
myfunction(x);
}
```

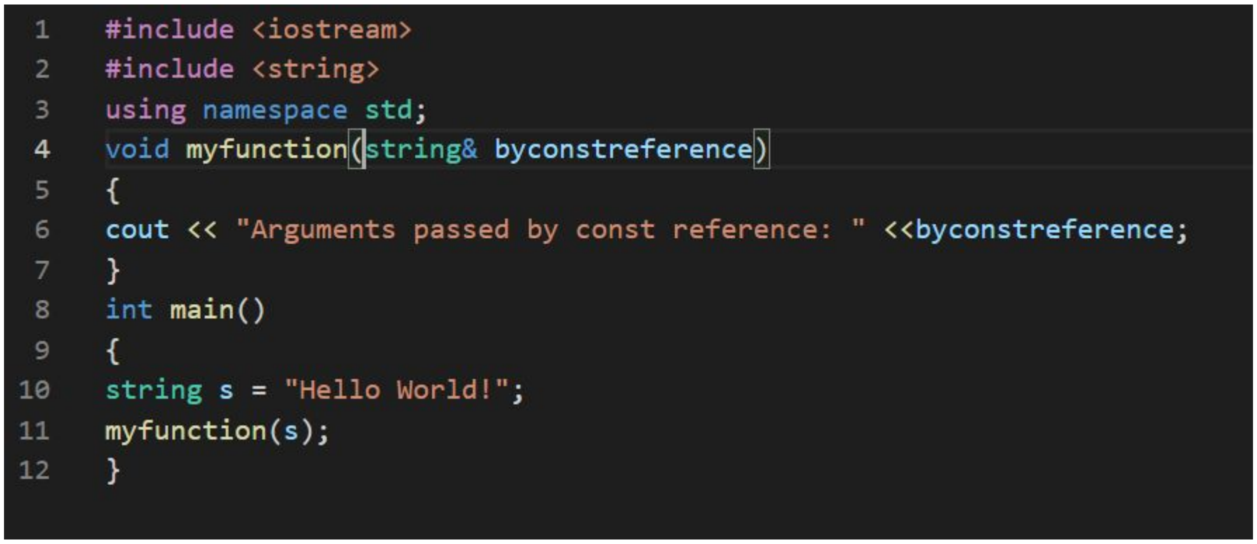
Output:



```
Argument passed by reference: 124
```

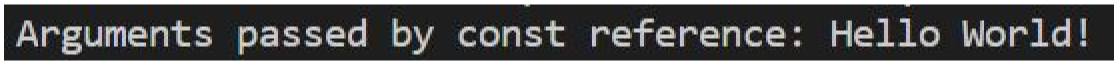
19.5.3 PASSING BY CONST REFERENCE

Example:



```
#include <iostream>
#include <string>
using namespace std;
void myfunction(string& byconstreference)
{
cout << "Arguments passed by const reference: " <<byconstreference;
}
int main()
{
string s = "Hello World!";
myfunction(s);
}
```

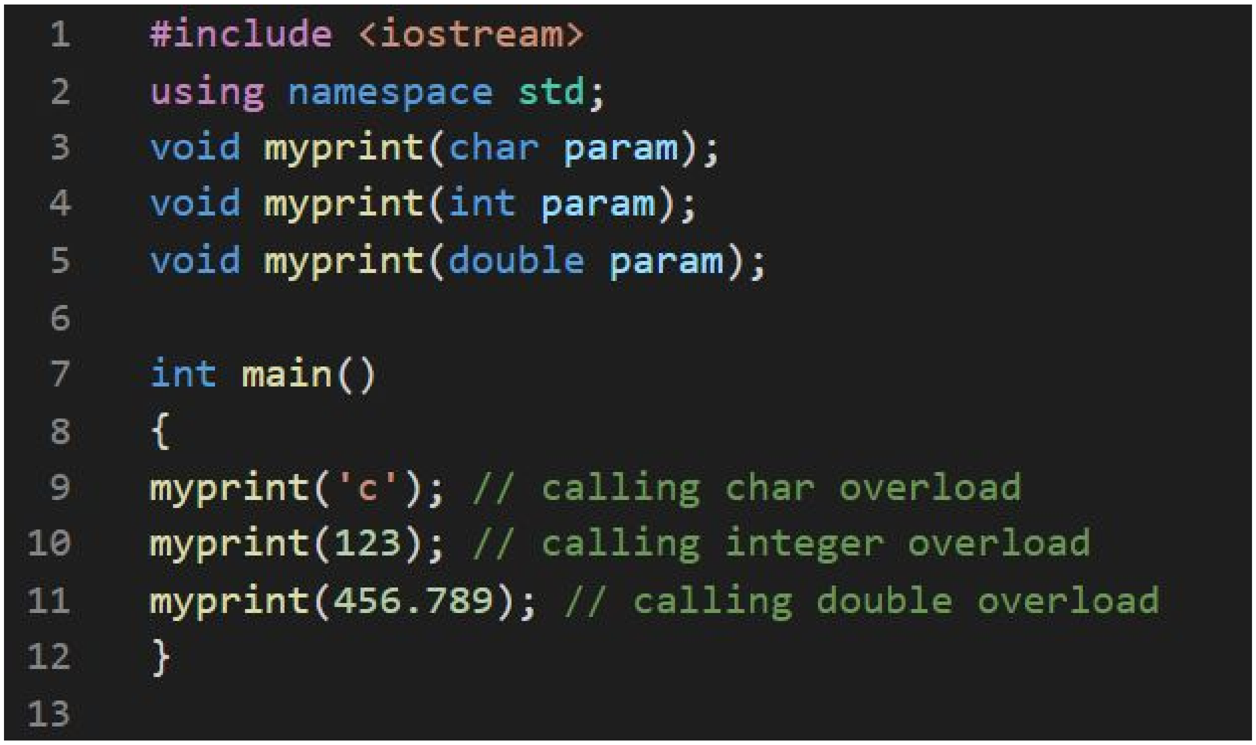
Output:



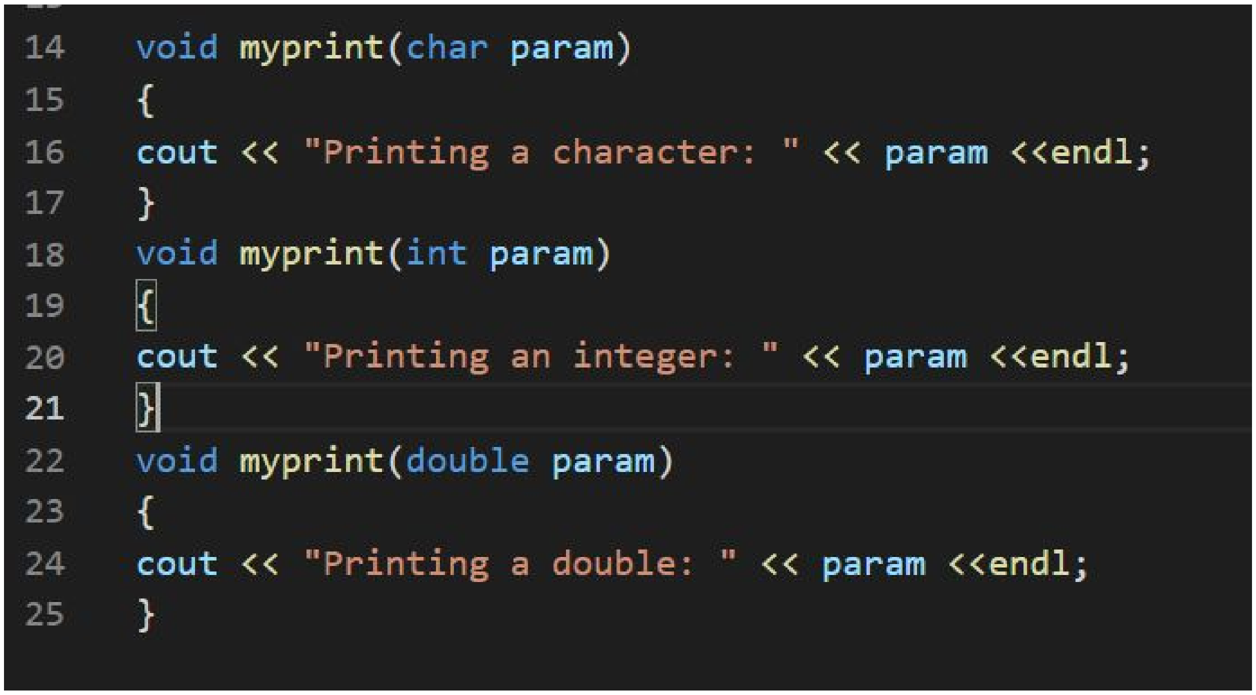
```
Arguments passed by const reference: Hello World!
```

19.6 FUNCTION OVERLOADING

- We can have multiple functions with the same name but with different parameter types. This is called function overloading. Example:

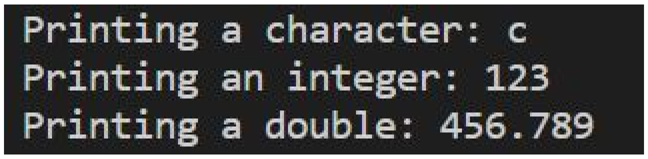


```
#include <iostream>
using namespace std;
void myprint(char param);
void myprint(int param);
void myprint(double param);

int main()
{
myprint('c'); // calling char overload
myprint(123); // calling integer overload
myprint(456.789); // calling double overload
}

```



```
void myprint(char param)
{
cout << "Printing a character: " << param <<endl;
}
void myprint(int param)
{
cout << "Printing an integer: " << param <<endl;
}
void myprint(double param)
{
cout << "Printing a double: " << param <<endl;
}
```

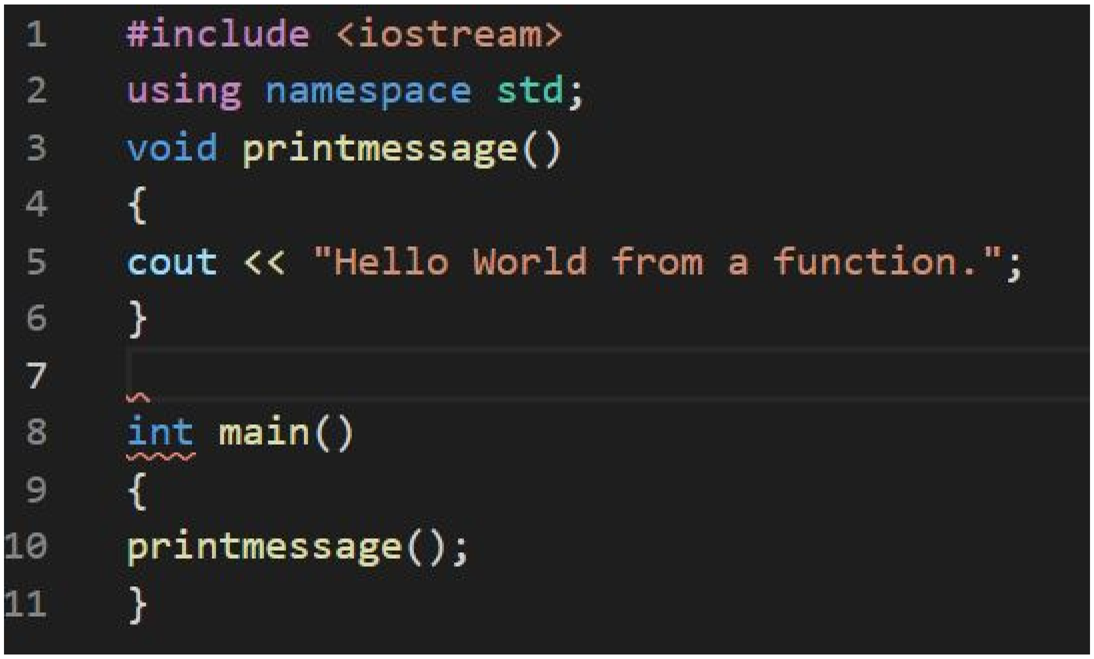
Output:



```
Printing a character: c
Printing an integer: 123
Printing a double: 456.789
```

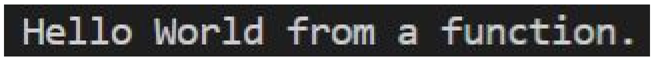
CHAPTER 20: EXERCISES

20.1 FUNCTION DEFINITION



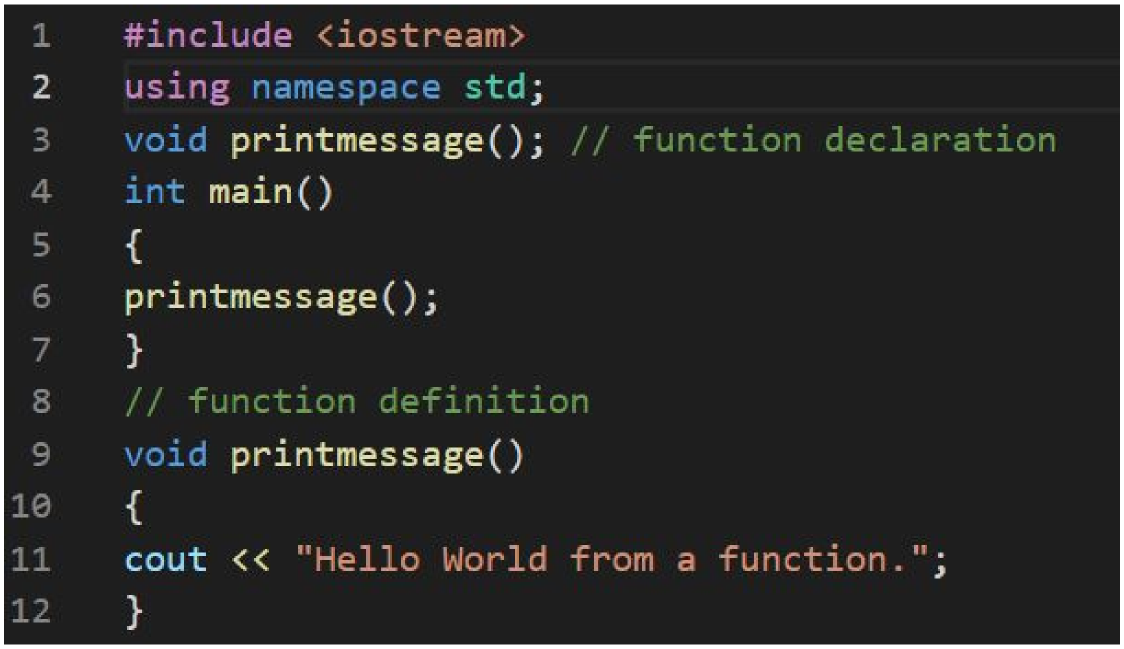
```cpp
#include <iostream>
using namespace std;
void printmessage()
{
cout << "Hello World from a function.";
}

int main()
{
printmessage();
}
```

Output:



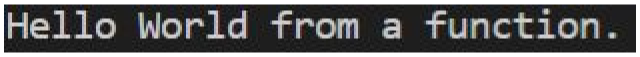
```
Hello World from a function.
```

20.2 SEPARATE DECLARATION AND DEFINITION



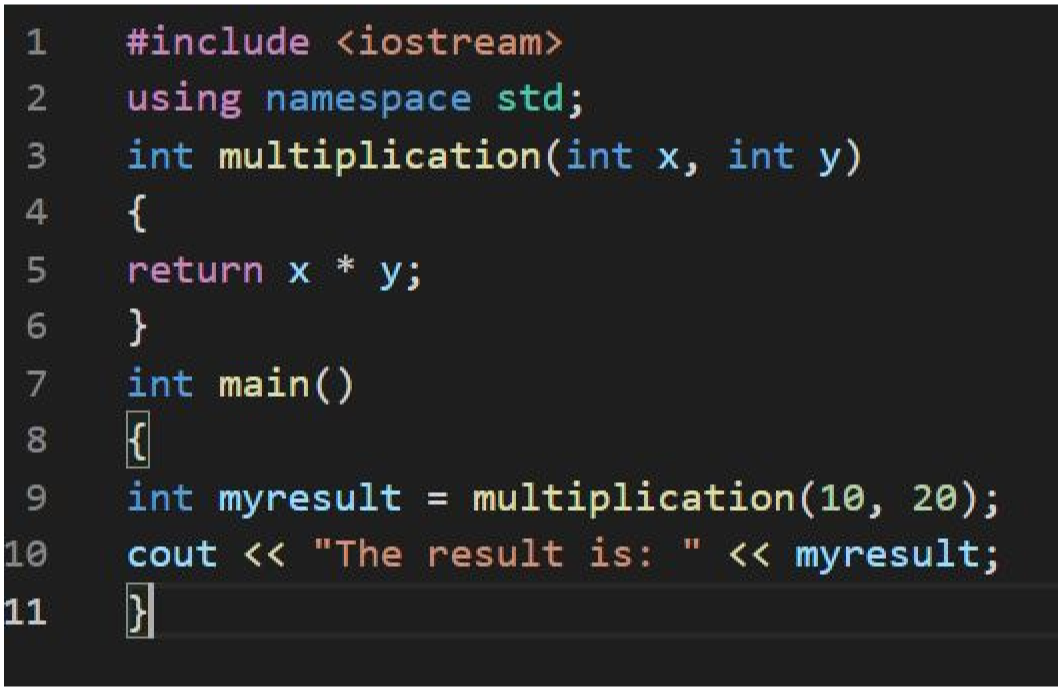
```cpp
#include <iostream>
using namespace std;
void printmessage(); // function declaration
int main()
{
printmessage();
}
// function definition
void printmessage()
{
cout << "Hello World from a function.";
}
```

Output:



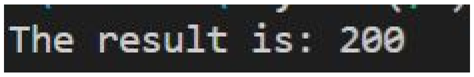
```
Hello World from a function.
```

20.3 FUNCTION PARAMETERS



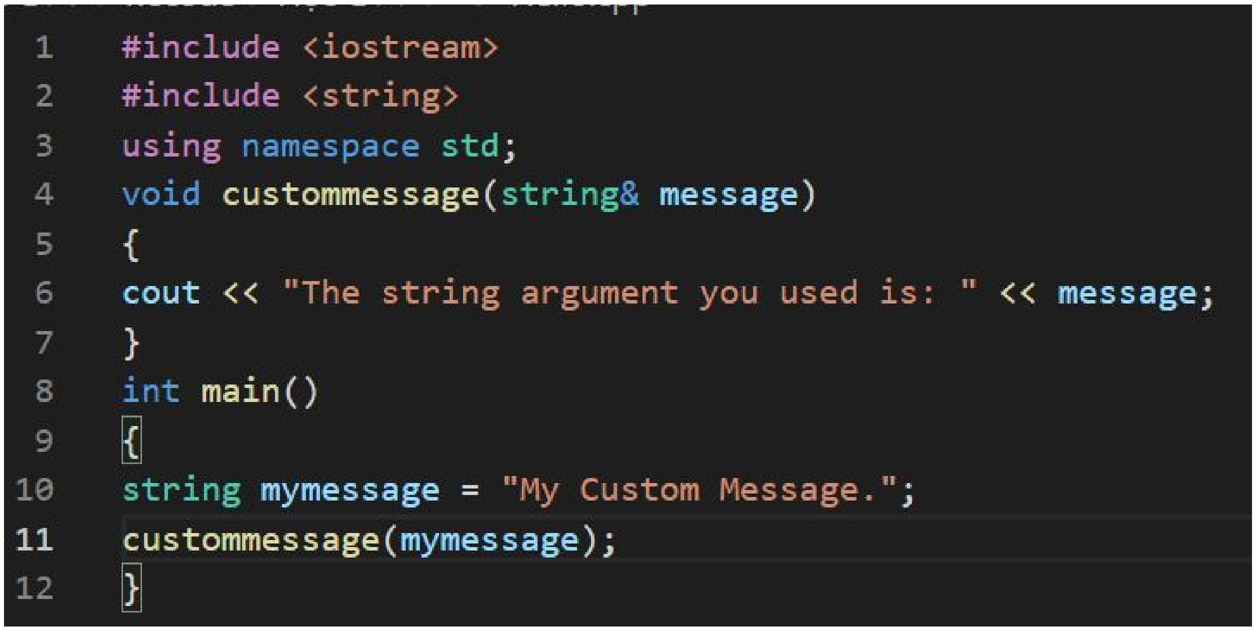
```cpp
#include <iostream>
using namespace std;
int multiplication(int x, int y)
{
return x * y;
}
int main()
{
int myresult = multiplication(10, 20);
cout << "The result is: " << myresult;
}
```

Output:



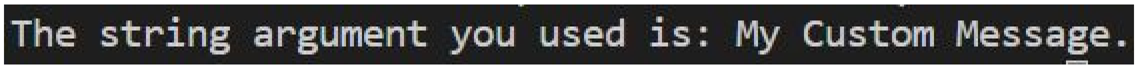
```
The result is: 200
```

20.4 PASSING ARGUMENT



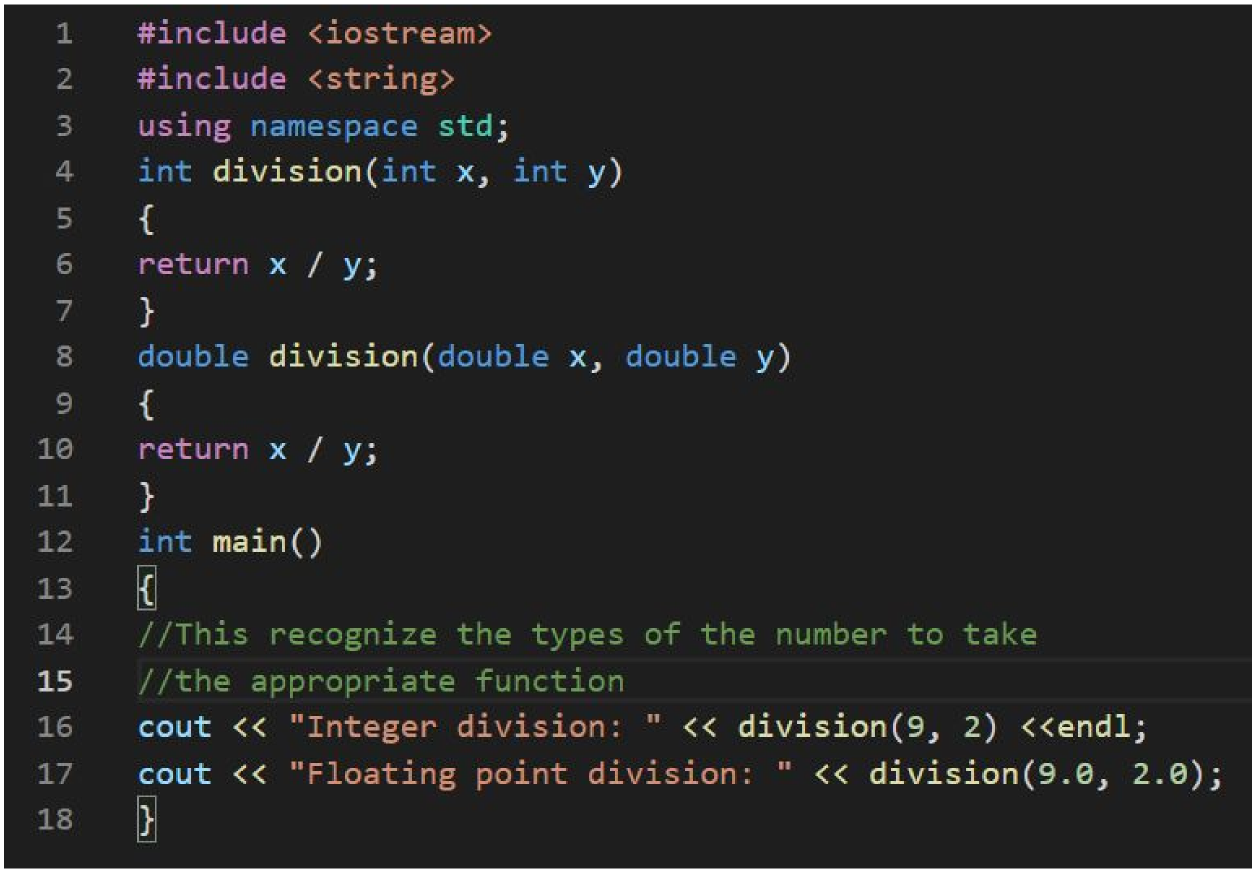
```cpp
#include <iostream>
#include <string>
using namespace std;
void custommessage(string& message)
{
cout << "The string argument you used is: " << message;
}
int main()
{
string mymessage = "My Custom Message.";
custommessage(mymessage);
}
```

Output:



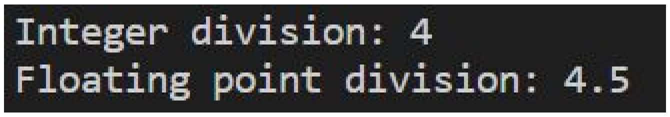
```
The string argument you used is: My Custom Message.
```

20.5 FUNCTION OVERLOADS



```cpp
#include <iostream>
#include <string>
using namespace std;
int division(int x, int y)
{
return x / y;
}
double division(double x, double y)
{
return x / y;
}
int main()
{
//This recognize the types of the number to take
//the appropriate function
cout << "Integer division: " << division(9, 2) <<endl;
cout << "Floating point division: " << division(9.0, 2.0);
}
```

Output:



```
Integer division: 4
Floating point division: 4.5
```