Lecture 04

M M Imran

Type Conversion

- Casting is when a value is changed from one data type to another.
- Only selected data types may be cast to other data types.
- Casting is either done:
  - Automatically by C++ – this is called implicit casting.
  - Coded by the programmer – this is called explicit casting.

	int expression →	float / double expression →	string expression →	char expression →
int variable	No cast is needed.	Implicit cast from float/double to int. Value is truncated.	Cannot cast.	Implicit cast from char to int. Value is converted to its ASCII / UNICODE code.
float / double variable	Implicit cast from int to float/double.	No cast is needed.	Cannot cast.	First, implicit cast from char to int. Value is converted to its ASCII / UNICODE code. Second, implicit cast from int to float/double.

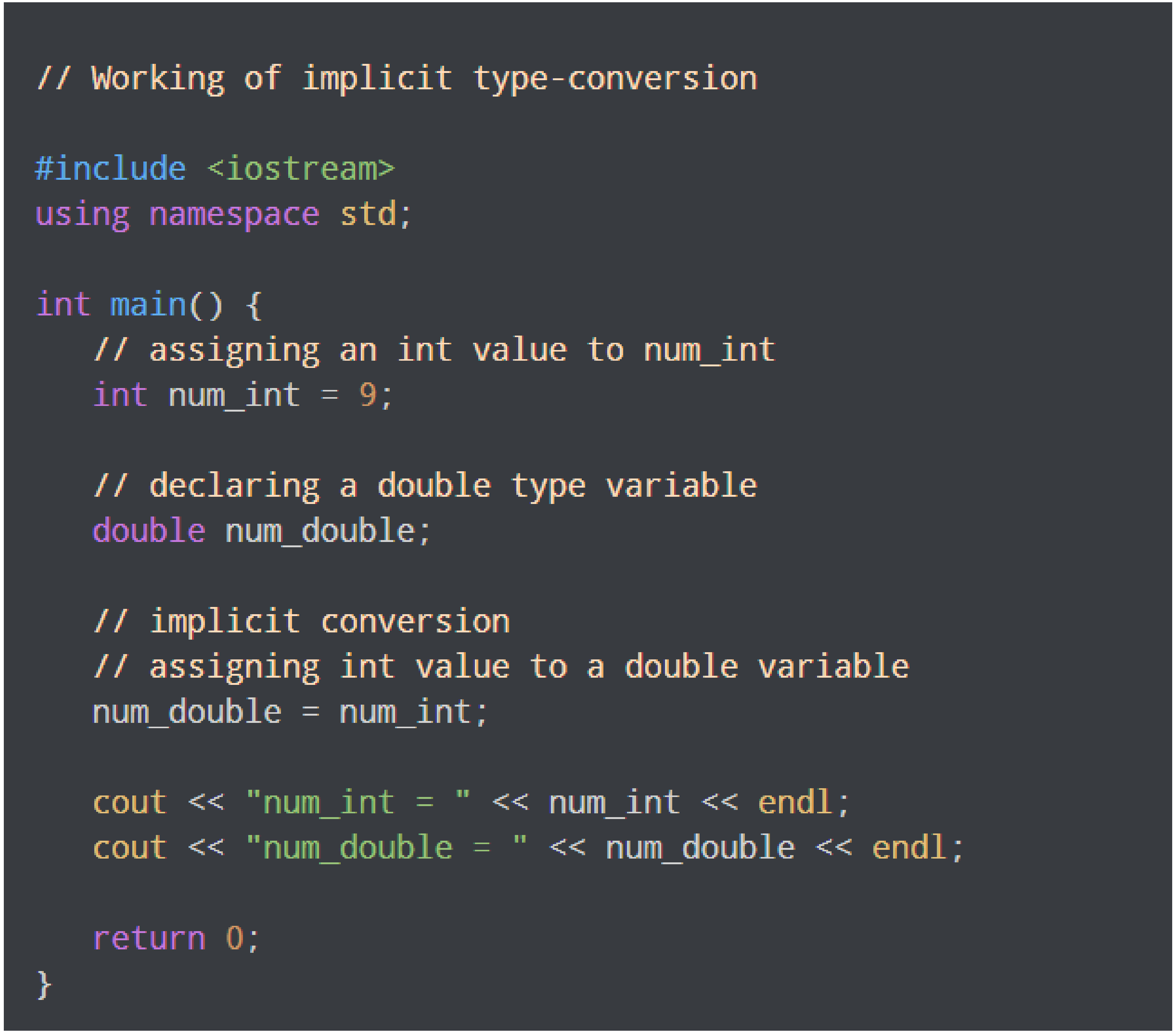
	int expression →	float / double expression →	string expression →	char expression →
string variable ←	*First, implicit cast from int to char. Value is converted from its ASCII / UNICODE code. Second, implicit cast from char to string.	*First, implicit cast from float/double to int. Value is truncated. Second, implicit cast from int to char. Value is converted from its ASCII / UNICODE code. Third, implicit cast from char to string.	No cast is needed.	Implicit cast from char to string. Value is converted to string.
char variable ←	*Implicit cast from int to char. Value is converted from its ASCII / UNICODE code.	*First, implicit cast from float/double to int. Value is truncated. Second, implicit cast from int to char. Value is converted from its ASCII / UNICODE code.	Cannot cast.	No cast is needed.

Explicit cast

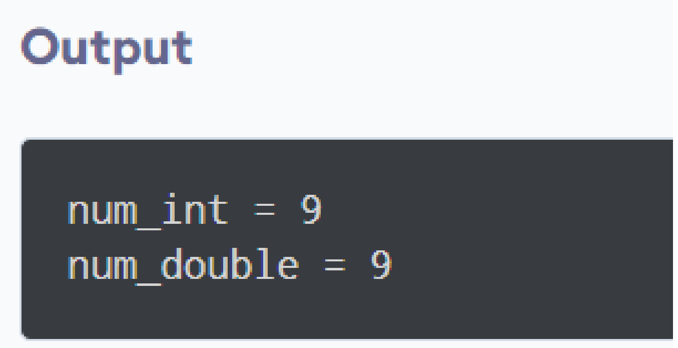
- To do an explicit cast on an expression, the data type to convert to is placed in parentheses before the expression.

Implicit Type Conversion

- The type conversion that is done automatically done by the compiler is known as implicit type conversion.
- This type of conversion is also known as automatic conversion.

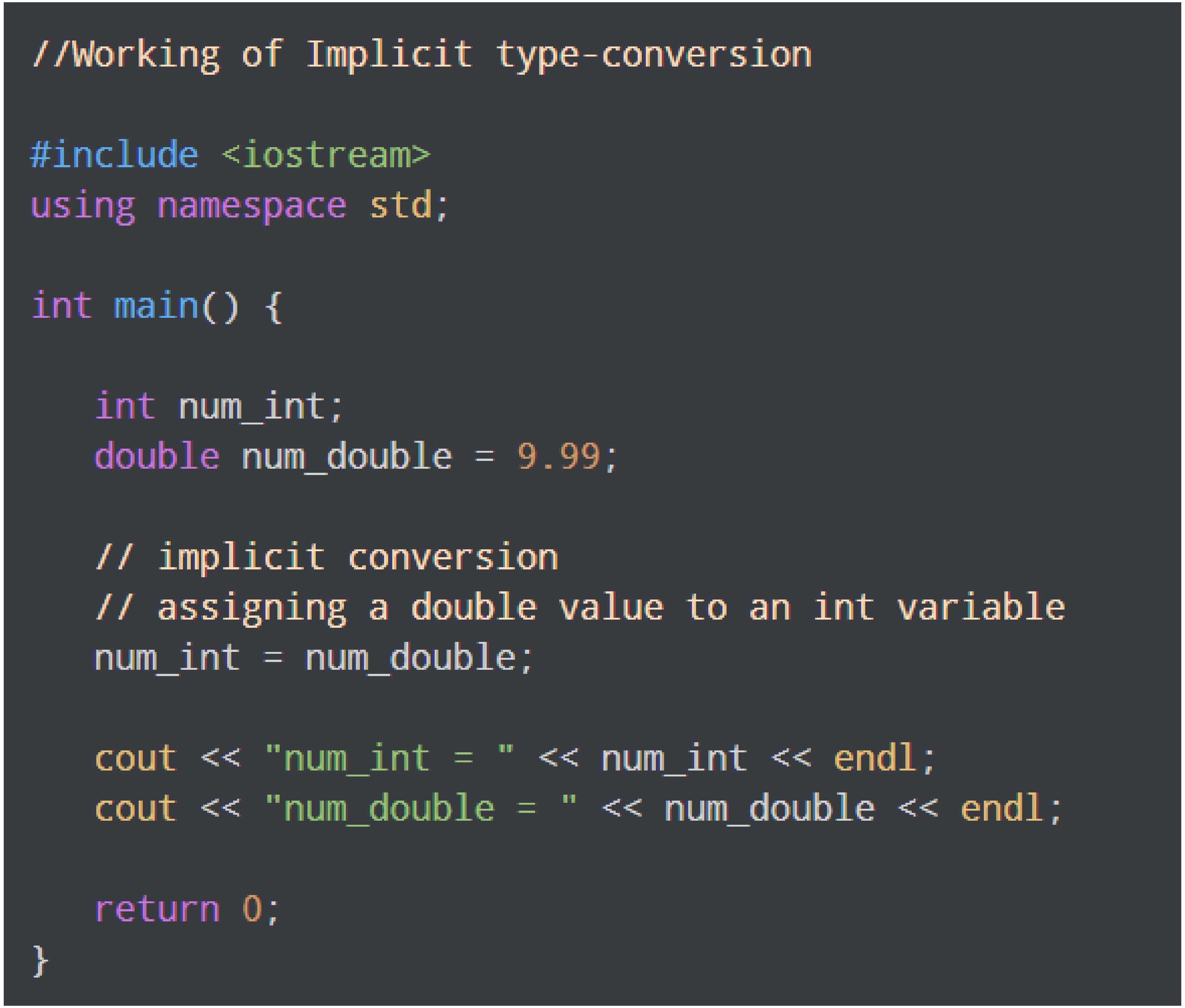


```cpp
// Working of implicit type-conversion

#include <iostream>
using namespace std;

int main() {
   // assigning an int value to num_int
   int num_int = 9;

   // declaring a double type variable
   double num_double;

   // implicit conversion
   // assigning int value to a double variable
   num_double = num_int;

   cout << "num_int = " << num_int << endl;
   cout << "num_double = " << num_double << endl;

   return 0;
}
```

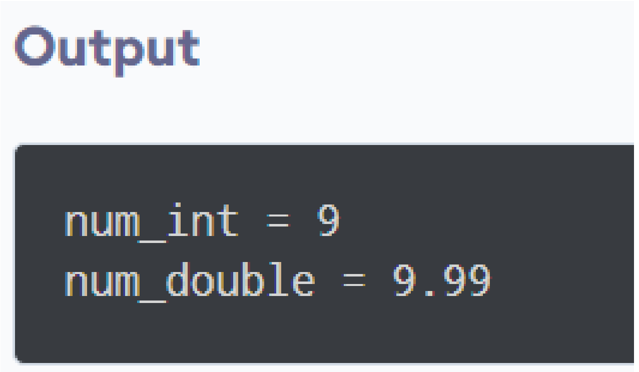


Output

```
num_int = 9
num_double = 9
```



```cpp
//Working of Implicit type-conversion

#include <iostream>
using namespace std;

int main() {

   int num_int;
   double num_double = 9.99;

   // implicit conversion
   // assigning a double value to an int variable
   num_int = num_double;

   cout << "num_int = " << num_int << endl;
   cout << "num_double = " << num_double << endl;

   return 0;
}
```

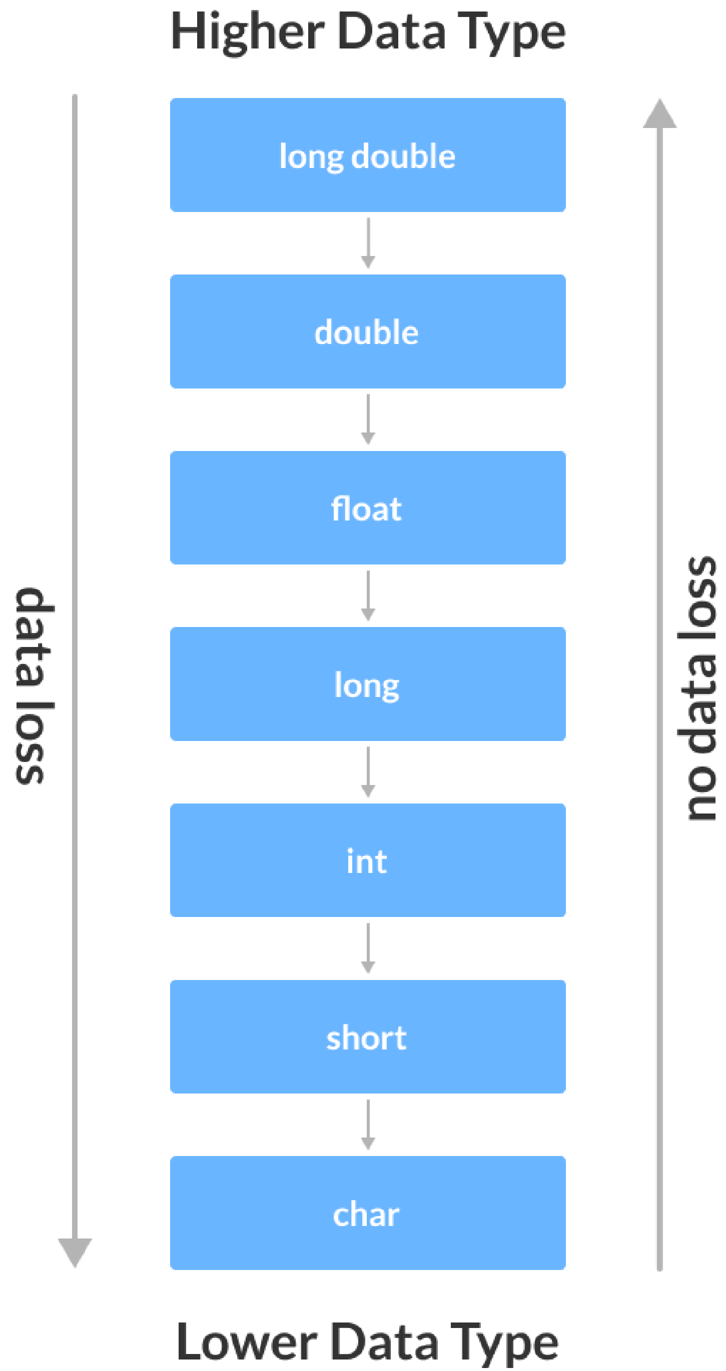


Output

```
num_int = 9
num_double = 9.99
```

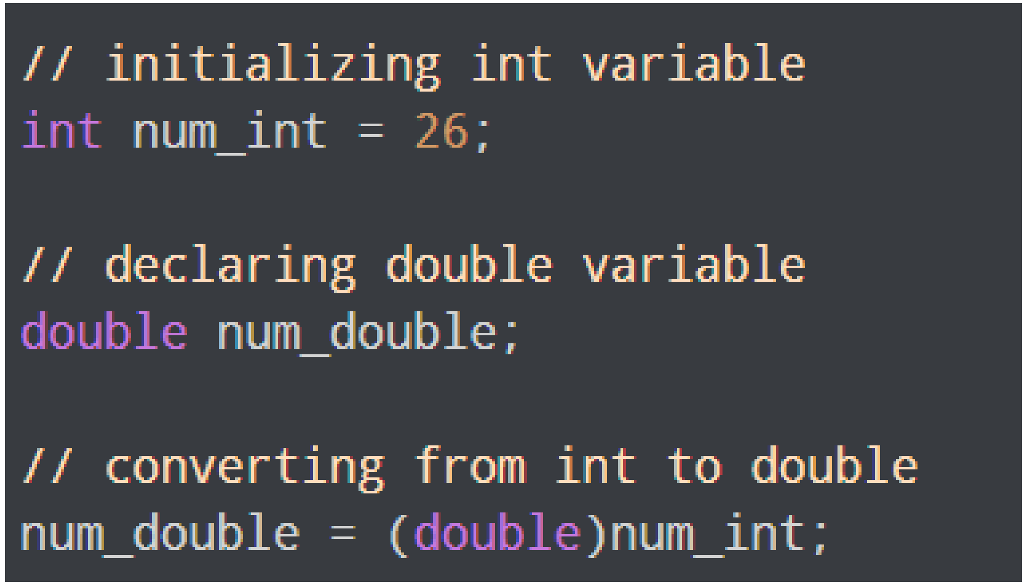
Data Loss During Conversion

- Conversion from one data type to another is prone to data loss.
- This happens when data of a larger type is converted to data of a smaller type.

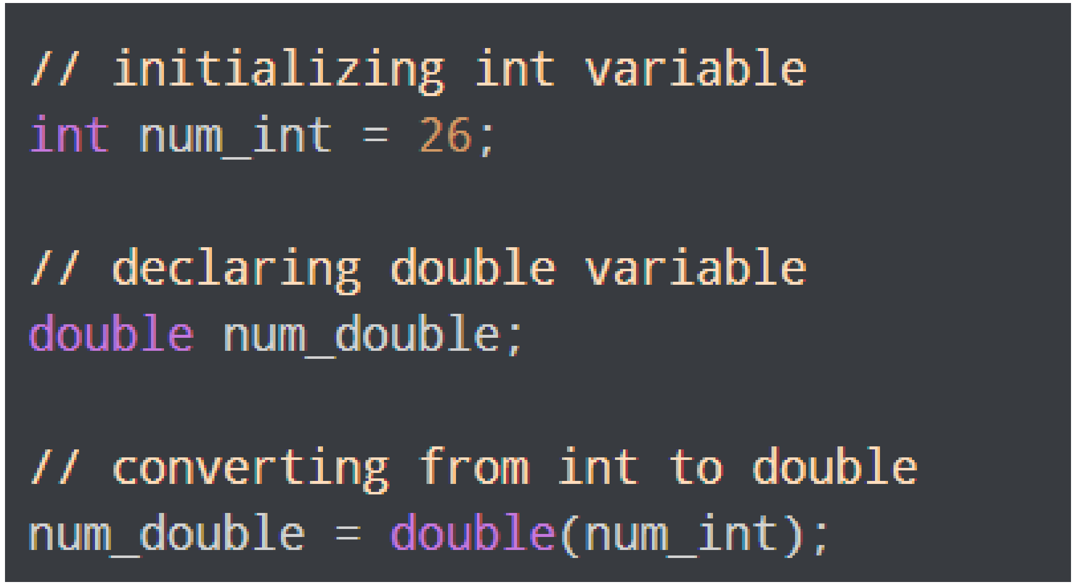


Explicit Type Conversion

- When the user manually changes data from one type to another, this is known as explicit conversion.
- This type of conversion is also known as type casting.
- The syntax for this style is:
  - **(data_type)expression;** // c-style conversion
  - **data_type(expression);** // function-style conversion



```
// initializing int variable
int num_int = 26;

// declaring double variable
double num_double;

// converting from int to double
num_double = (double)num_int;
```



```
// initializing int variable
int num_int = 26;

// declaring double variable
double num_double;

// converting from int to double
num_double = double(num_int);
```

Task

- Create a double variable **num_double**
- Create an integer variable **num_int1** from **num_double** using c style type conversion
- Create an integer variable **num_int1** from **num_double** using function style type conversion
- Print out all three variables

Type Conversion

- Assigning of non-bool to a bool variable yields false if the value is 0 and true otherwise.

```
bool b = 42;          // b is true
```

- Assigning of a bool to one of the other arithmetic types yields 1 if the bool is true and 0 if the bool is false.

```
bool b = false;
int i = b;            // i has value 0
```

Constants

- A constant is identical to a variable except its initial value cannot be changed while a program is running.
- A constant enables a value to be assigned a name which makes code easier to read.
- A constant enables a value, which may be used in many places in a program, to be changed in one place before the program runs.
- A constant has at least three characteristics:
    - Name
    - Data type
    - Value

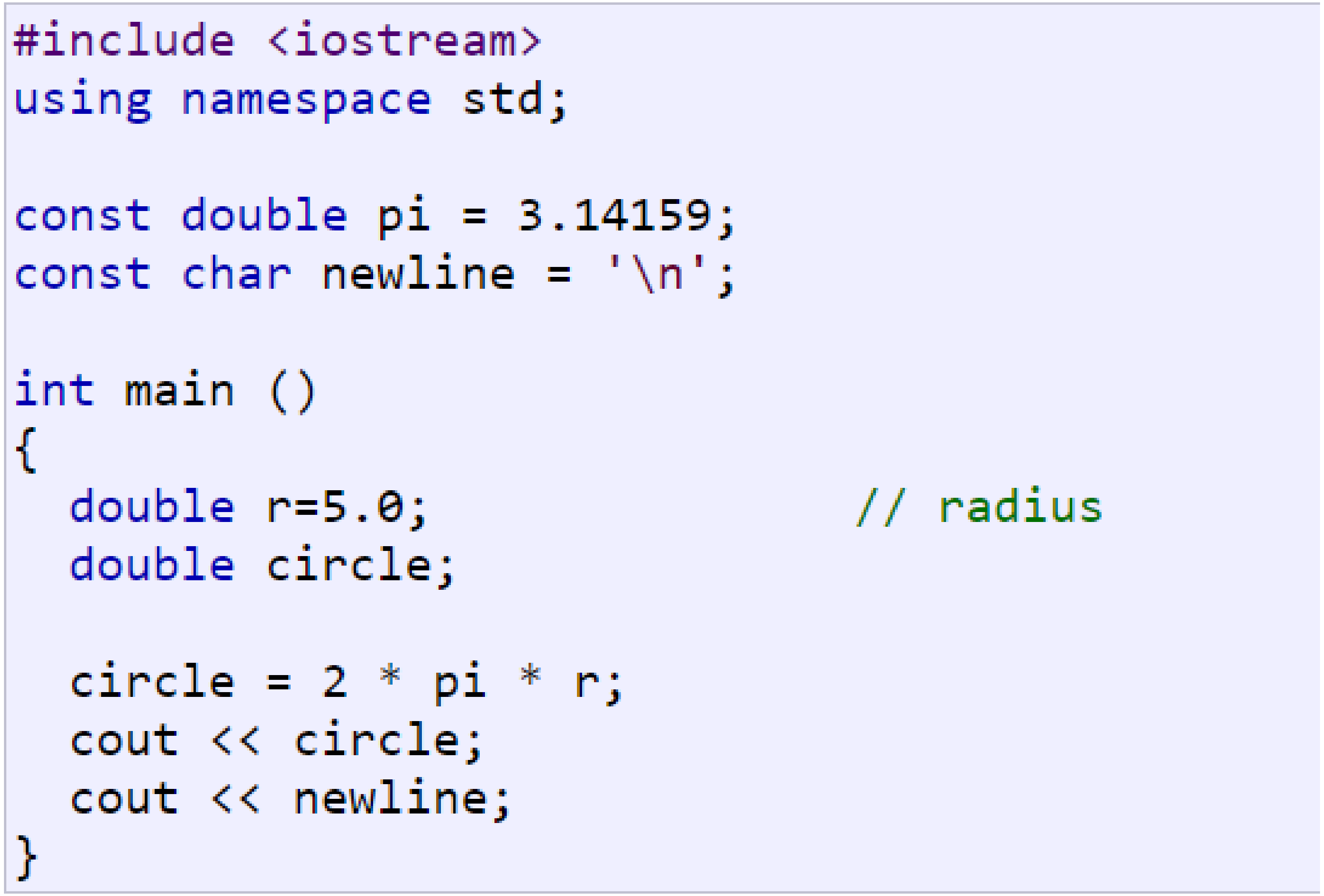
Constant Declaration

- There are two methods to define constants in C++.
  - #define preprocessor directive method

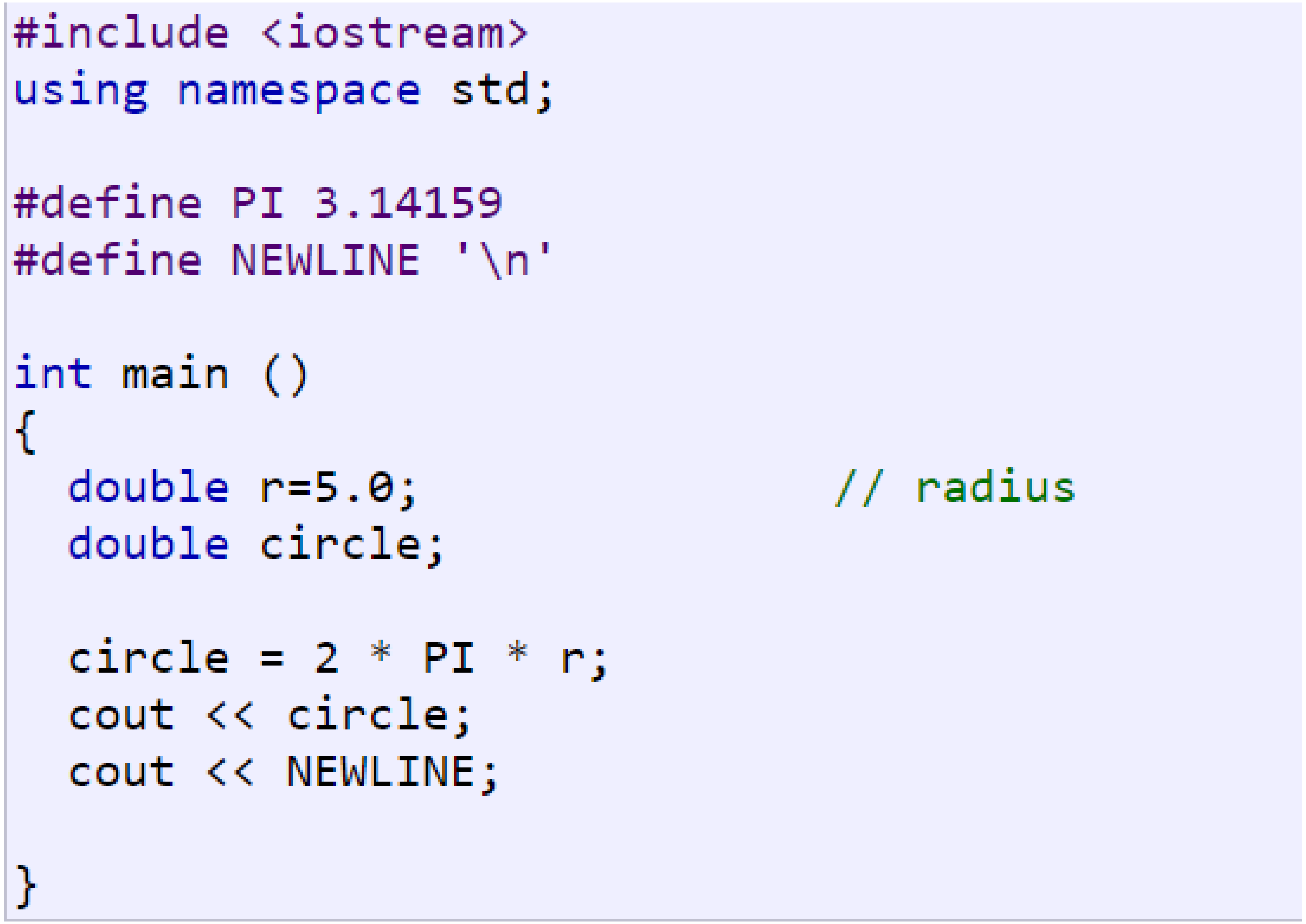
    #define constantName value

  - const keyword method

    const datatype constantName = value;



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

const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;               // radius
  double circle;

  circle = 2 * pi * r;
  cout << circle;
  cout << newline;
}
```



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

#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
  double r=5.0;               // radius
  double circle;

  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;

}
```

Input Output Streams

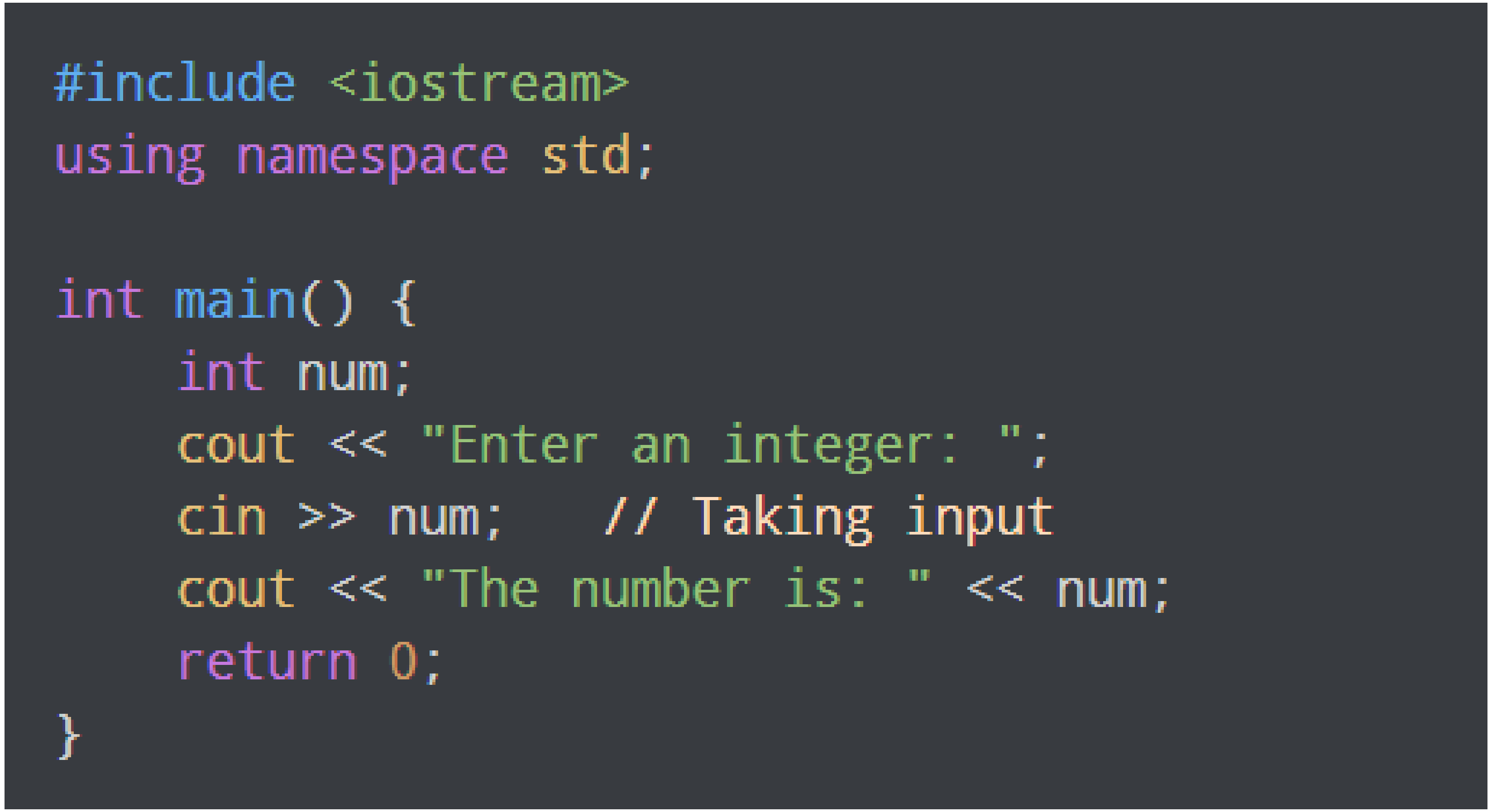
- A stream is a sequence of binary or character data.
- A stream goes in one of two directions:
  - Data entering an application, often from a keyboard, mouse, or file.
  - Data leaving an application, often to a screen or file.
- Library iostream is used to communicate with input and output devices.
- Common input (cin) is an object variable of type istream and is used to connect to the keyboard.
- Common output (cout) is an object variable of type ostream and is used to connect to the screen.
- To connect to the iostream library, the following statement is used in an application: **#include <iostream>**

Useful Header Files

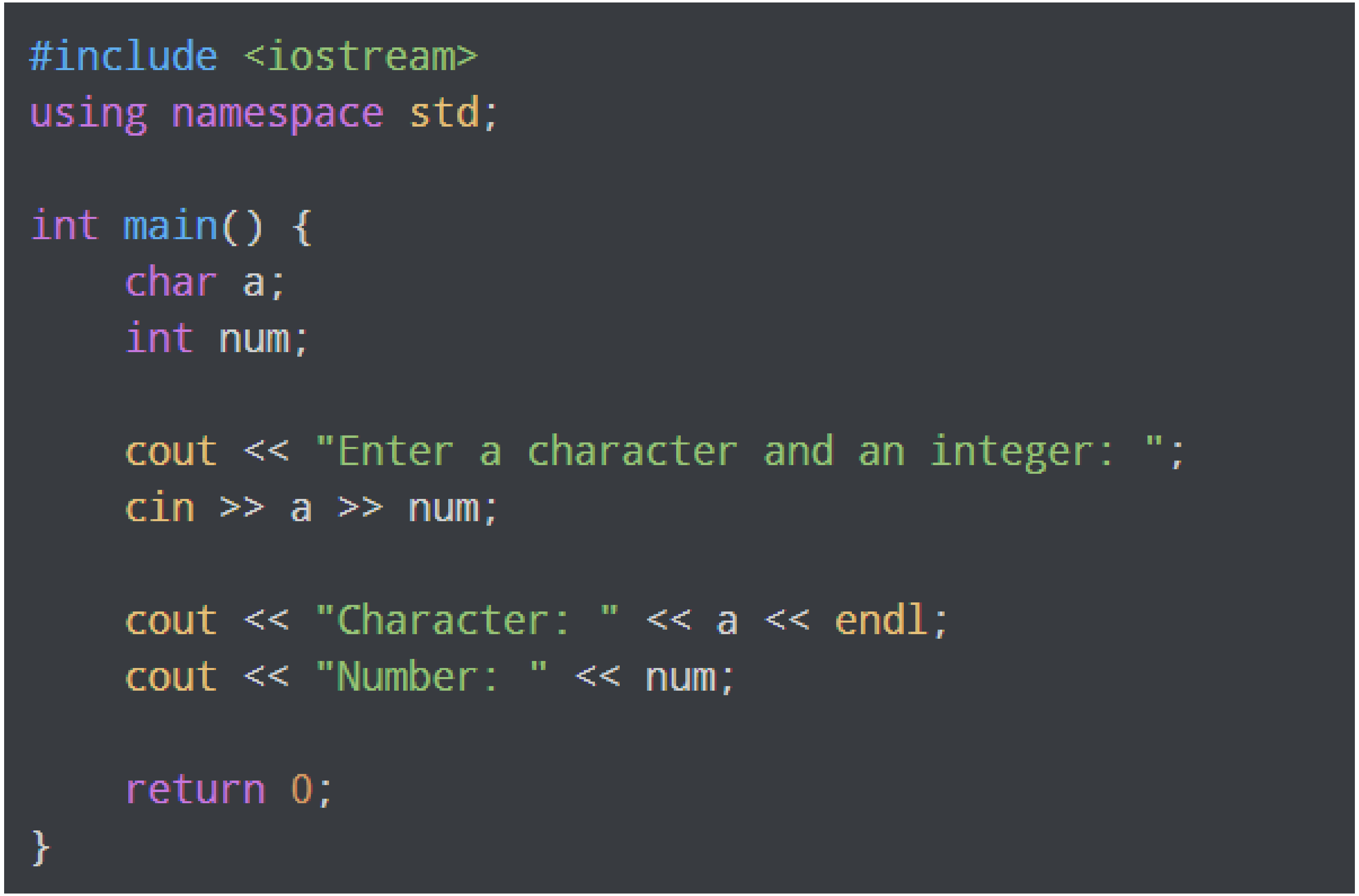
Header File	Function **and** Description
<iostream>	It is used to define the **cout, cin and cerr** objects, which correspond to standard output stream, standard input stream and standard error stream, respectively.
<iomanip>	It is used to declare services useful for performing formatted I/O, such as **setprecision and setw.**
<fstream>	It is used to declare services for user-controlled file processing.

Common Out (cout)

- **cout** writes data to the screen.
- **cout** combines one or more expressions.
- The expressions may be of type string, number, character, etc.
- **cout** expressions are separated by the insertion operator (<<).
- cout syntax: **cout << <expression>[<< <expression>]*;**
- **endl** is one of several manipulators (it moves the screen cursor to the start of the next line).



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

int main() {
    int num;
    cout << "Enter an integer: ";
    cin >> num;   // Taking input
    cout << "The number is: " << num;
    return 0;
}
```



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

int main() {
    char a;
    int num;

    cout << "Enter a character and an integer: ";
    cin >> a >> num;

    cout << "Character: " << a << endl;
    cout << "Number: " << num;

    return 0;
}
```

Escape Sequences

- An escape sequence is a pair of characters used in a string to perform one of two operations:
  - Enable a special character to be outputted.
  - Move the output cursor.
- An escape sequence is included within a string literal.
- The first character is the backslash (\).
- Several escape sequences are available for use in the cout statement

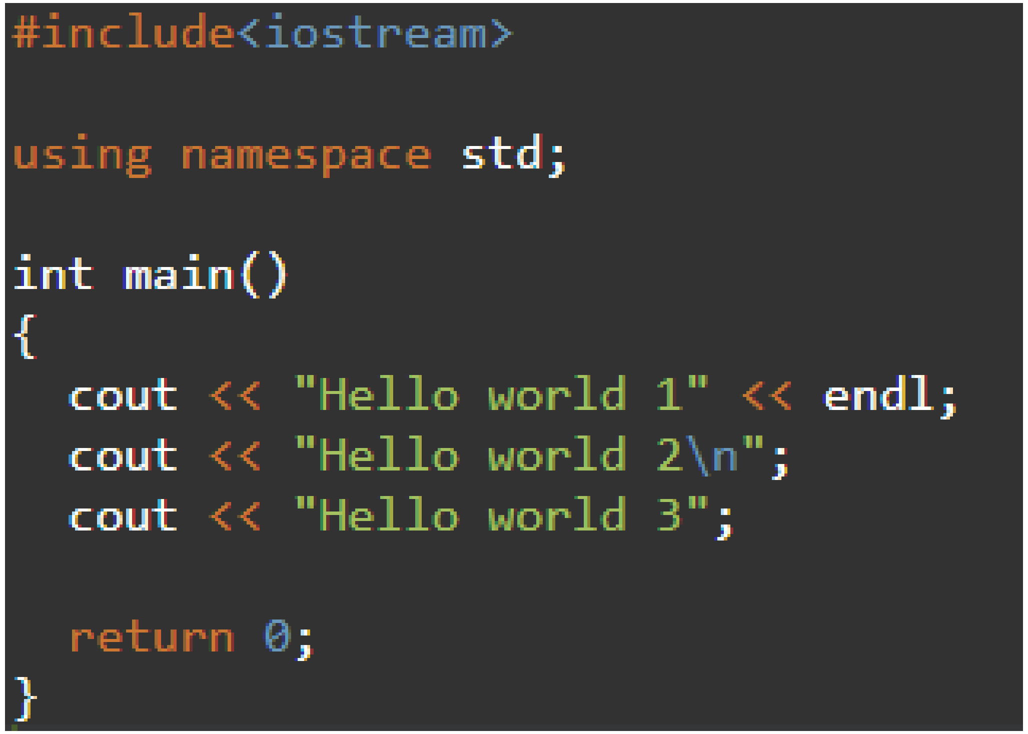
Sequence	Purpose
\\	Output a backslash.
\’	Output a single quote.
\”	Output a double quote.
\a	Beep user.
\b	Backspace cursor.
\n	Move cursor to start of next line.
\t	Tab on the current line.

Formatted Output

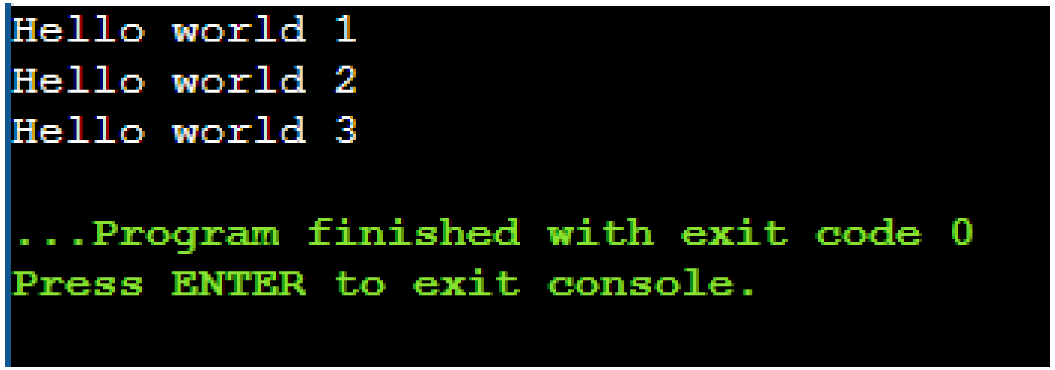
- Formatting in the standard C++ libraries is done through the use of manipulators, special variables or objects that are placed on the output stream.
- Most of the standard manipulators are found in <iostream> and so are included automatically.

endl

- Places a new line character on the output stream. This is identical to placing '\n' on the output stream.



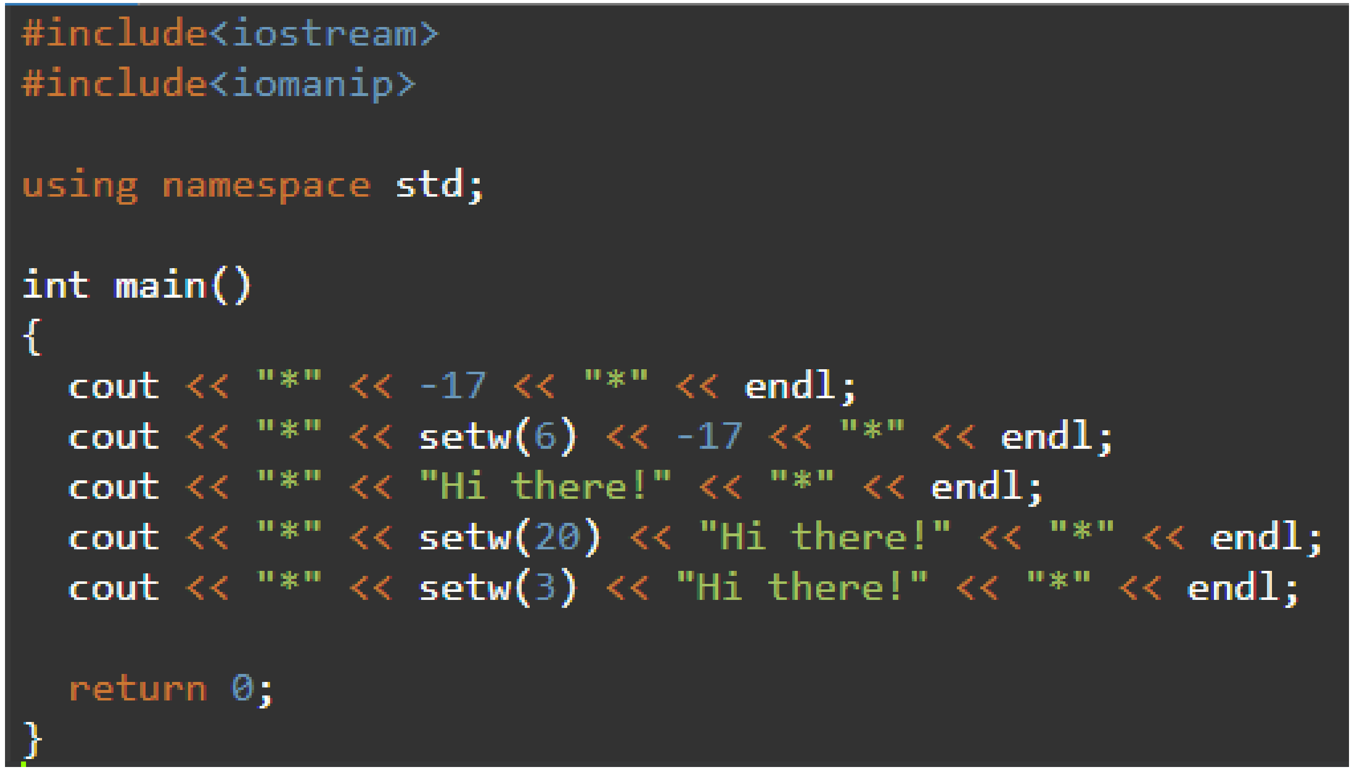
```
#include<iostream>

using namespace std;

int main()
{
  cout << "Hello world 1" << endl;
  cout << "Hello world 2\n";
  cout << "Hello world 3";

  return 0;
}
```



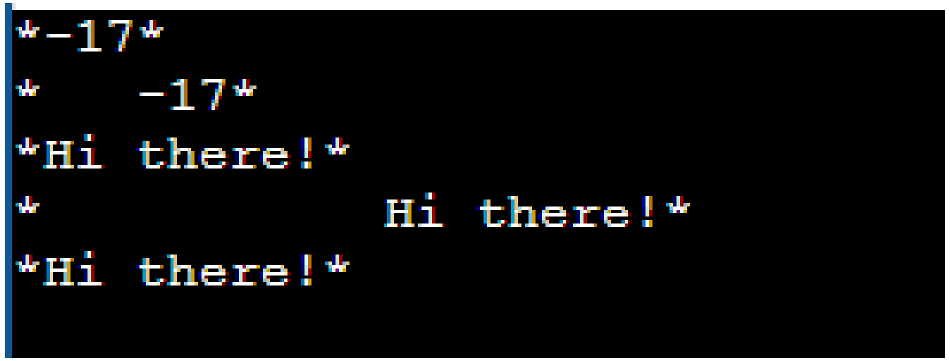
```
Hello world 1
Hello world 2
Hello world 3

...Program finished with exit code 0
Press ENTER to exit console.
```

setw()

- Adjusts the field width with for the item about to be printed.
- Needs <iomanip> header



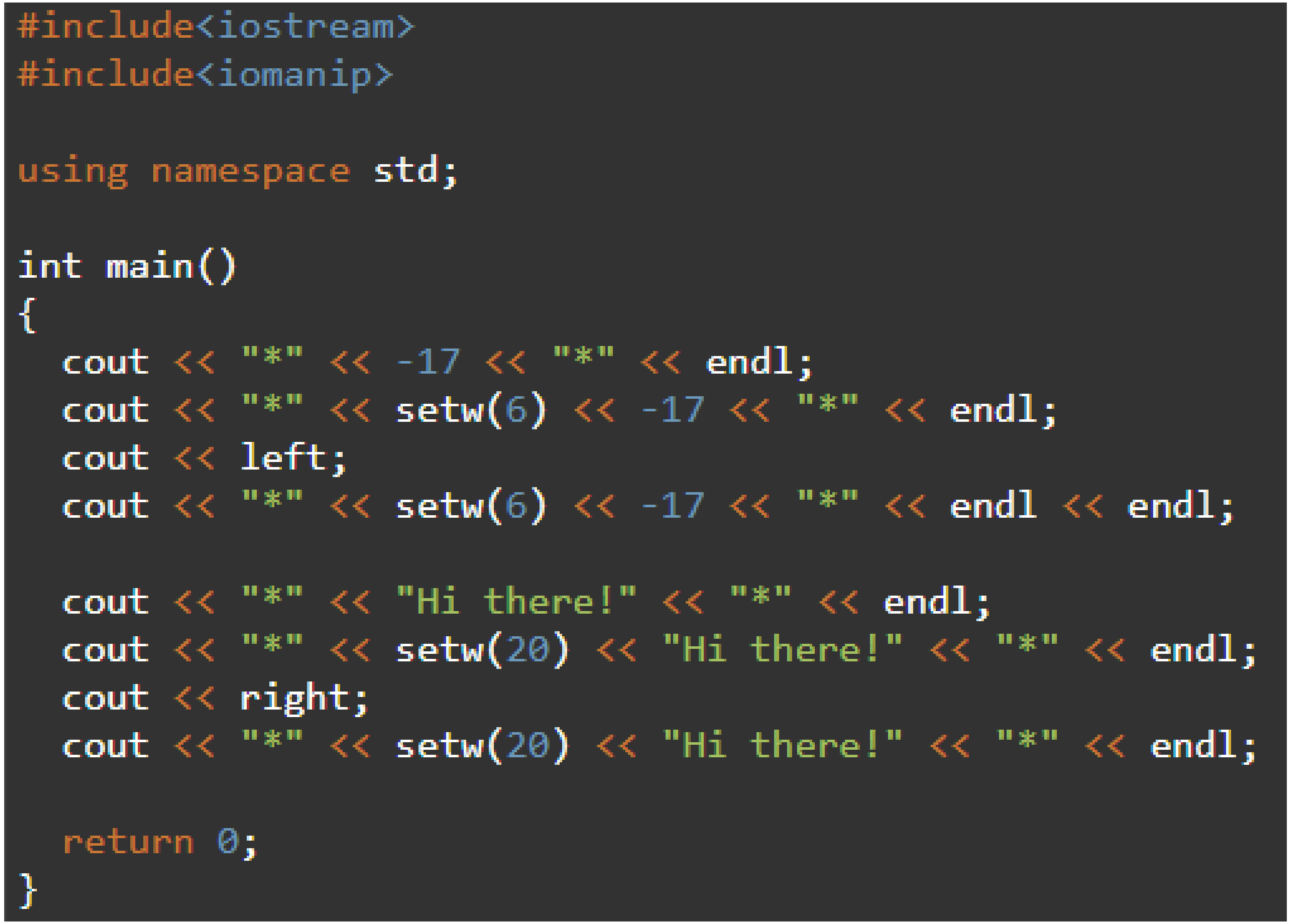
```cpp
#include<iostream>
#include<iomanip>

using namespace std;

int main()
{
  cout << "*" << -17 << "*" << endl;
  cout << "*" << setw(6) << -17 << "*" << endl;
  cout << "*" << "Hi there!" << "*" << endl;
  cout << "*" << setw(20) << "Hi there!" << "*" << endl;
  cout << "*" << setw(3) << "Hi there!" << "*" << endl;

  return 0;
}
```



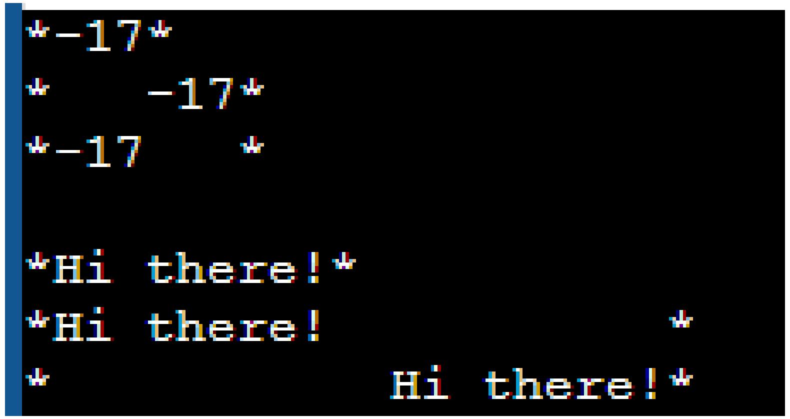
```
*-17*
*   -17*
*Hi there!*
*           Hi there!*
*Hi there!*
```

left and **right**

- **left**: left justify all values in their fields.
- **right**: right justify all values in their fields. This is the default justification value.



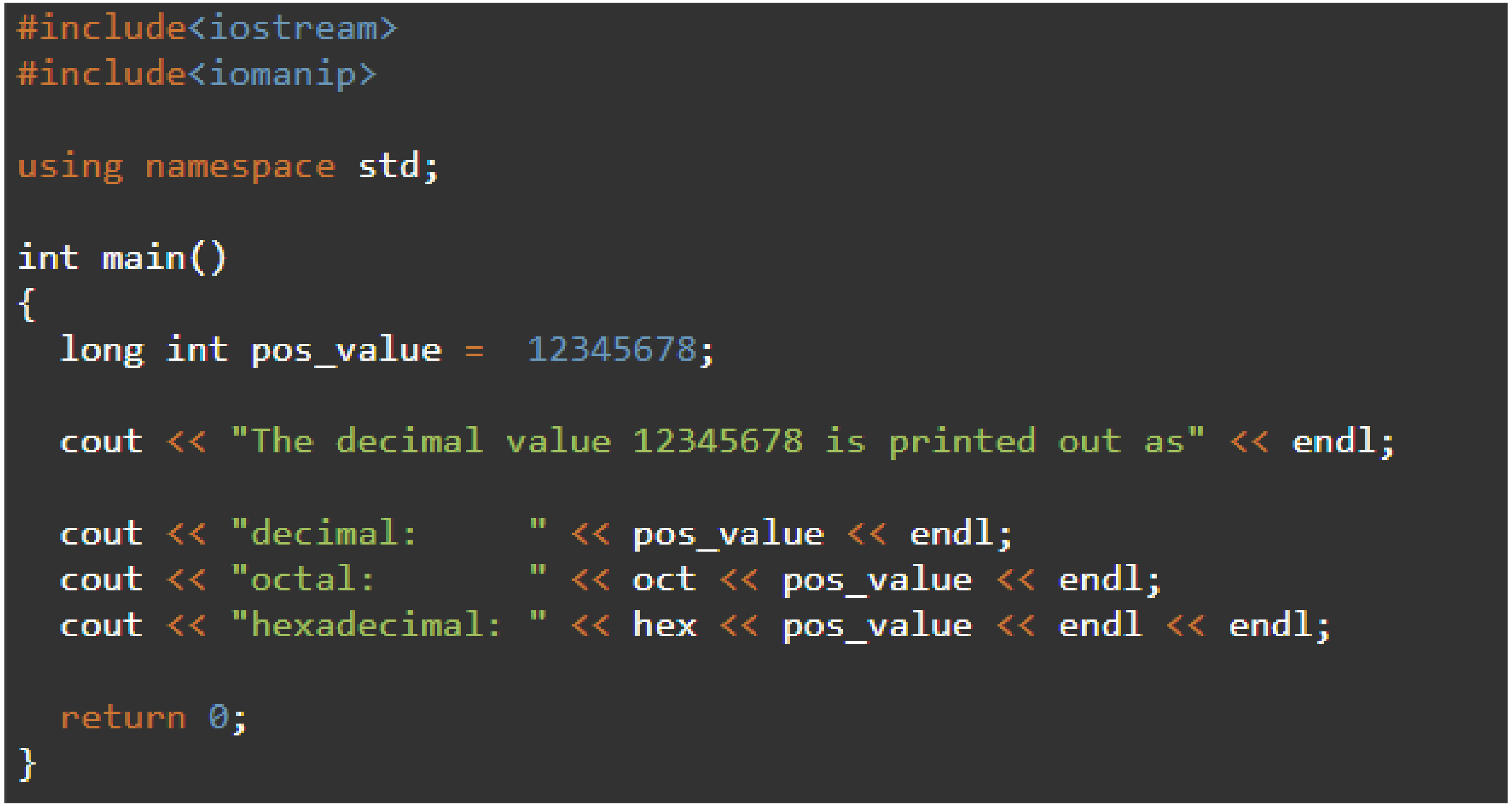
```cpp
#include<iostream>
#include<iomanip>

using namespace std;

int main()
{
   cout << "*" << -17 << "*" << endl;
   cout << "*" << setw(6) << -17 << "*" << endl;
   cout << left;
   cout << "*" << setw(6) << -17 << "*" << endl << endl;

   cout << "*" << "Hi there!" << "*" << endl;
   cout << "*" << setw(20) << "Hi there!" << "*" << endl;
   cout << right;
   cout << "*" << setw(20) << "Hi there!" << "*" << endl;

   return 0;
}
```



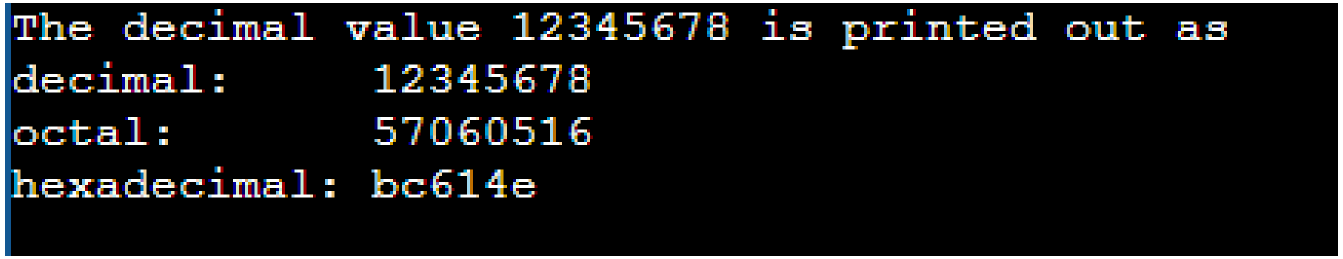
```
*-17*
*   -17*
*-17   *

*Hi there!*
*Hi there!           *
*           Hi there!*
```

Base

- The manipulators dec, oct, and hex change the base that is used to print out integer values.



```cpp
#include<iostream>
#include<iomanip>

using namespace std;

int main()
{
  long int pos_value =  12345678;

  cout << "The decimal value 12345678 is printed out as" << endl;

  cout << "decimal:     " << pos_value << endl;
  cout << "octal:       " << oct << pos_value << endl;
  cout << "hexadecimal: " << hex << pos_value << endl << endl;

  return 0;
}
```

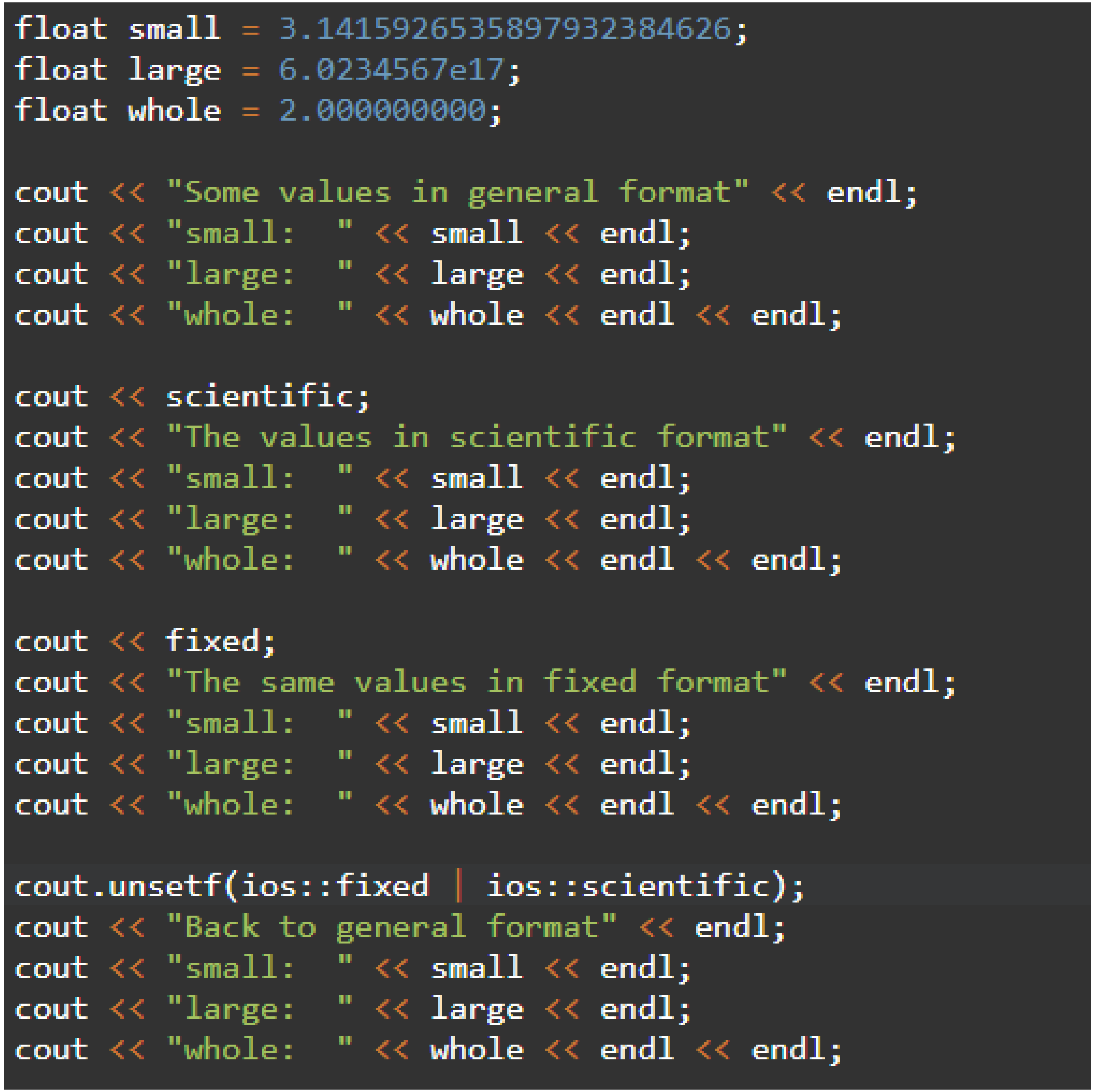


```
The decimal value 12345678 is printed out as
decimal:     12345678
octal:       57060516
hexadecimal: bc614e
```

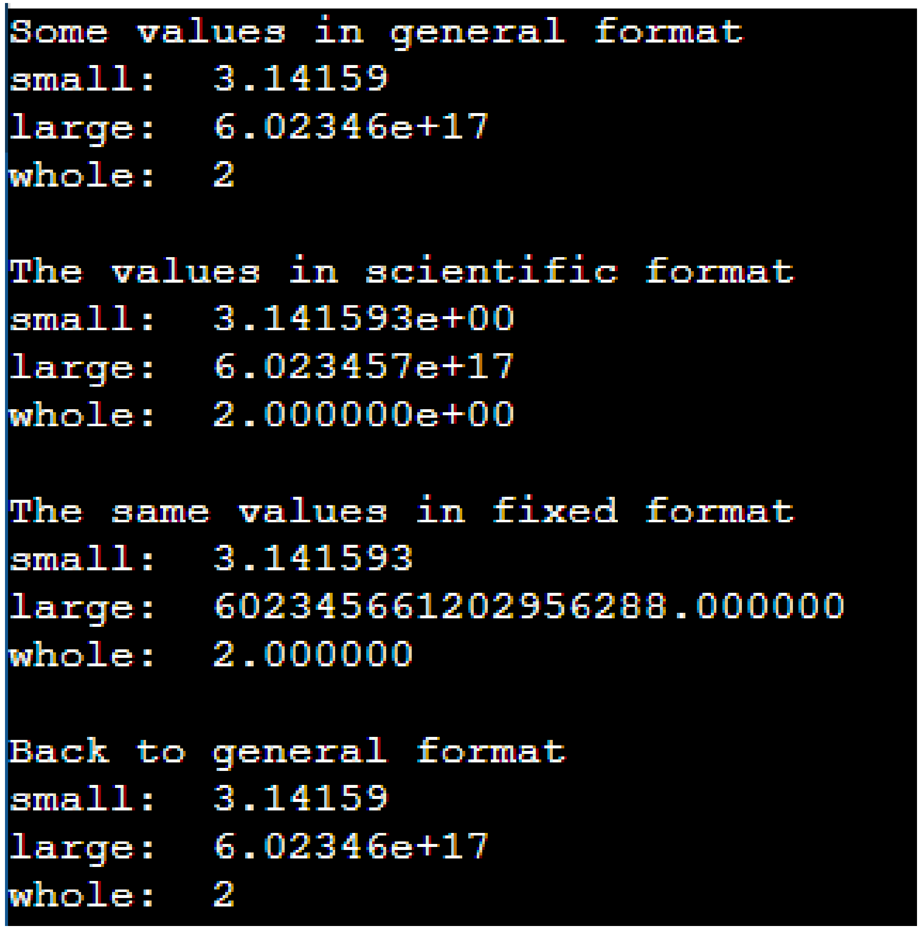
fixed and **scientific**

- The manipulator **fixed** will set up the output stream for displaying floating point values in fixed format.
- The **scientific** manipulator forces all floating point values to be displayed in scientific notation.
- To return back to normal form use:

```
cout.unsetf(ios::fixed | ios::scientific);
```



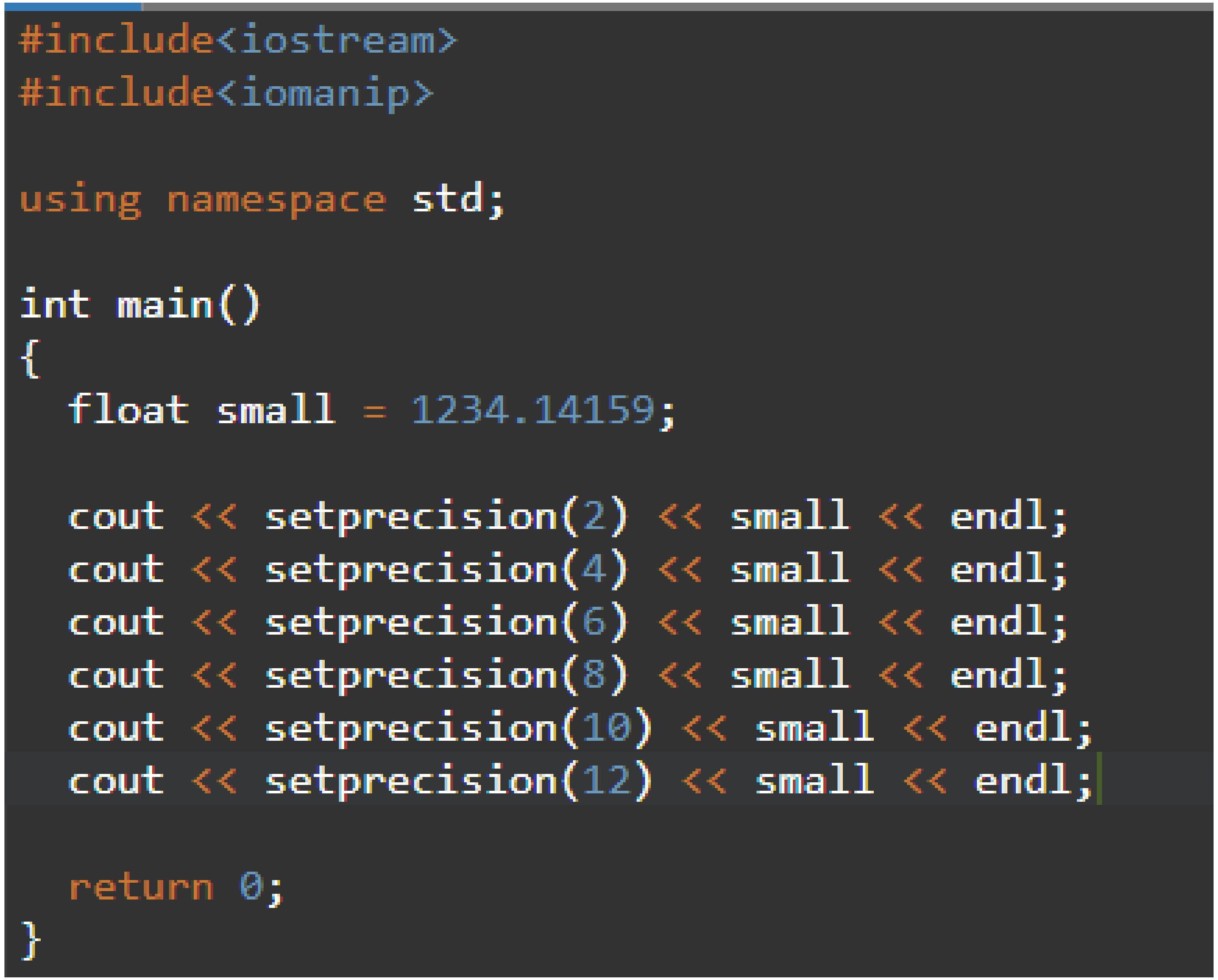
```cpp
float small = 3.14159265358979323846 26;
float large = 6.0234567e17;
float whole = 2.000000000;

cout << "Some values in general format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;

cout << scientific;
cout << "The values in scientific format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;

cout << fixed;
cout << "The same values in fixed format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;

cout.unsetf(ios::fixed | ios::scientific);
cout << "Back to general format" << endl;
cout << "small:  " << small << endl;
cout << "large:  " << large << endl;
cout << "whole:  " << whole << endl << endl;
```



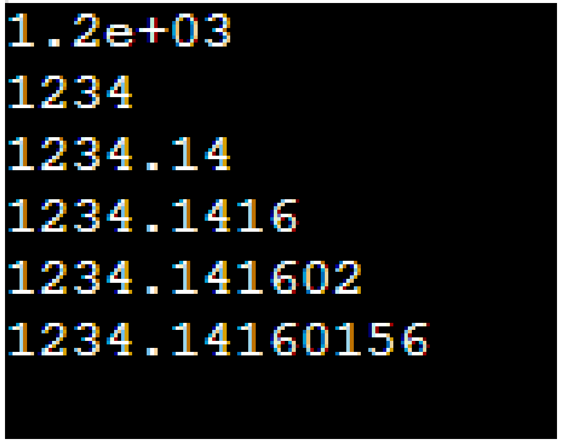
```
Some values in general format
small:  3.14159
large:  6.02346e+17
whole:  2

The values in scientific format
small:  3.141593e+00
large:  6.023457e+17
whole:  2.000000e+00

The same values in fixed format
small:  3.141593
large:  602345661202956288.000000
whole:  2.000000

Back to general format
small:  3.14159
large:  6.02346e+17
whole:  2
```

setprecision()

- Sets the decimal precision to be used to format floating-point values on output operations.
- The precision is the maximum number of digits displayed
- This includes digits before and after the decimal point, but does not include the decimal point itself.



```cpp
#include<iostream>
#include<iomanip>

using namespace std;

int main()
{
  float small = 1234.14159;

  cout << setprecision(2) << small << endl;
  cout << setprecision(4) << small << endl;
  cout << setprecision(6) << small << endl;
  cout << setprecision(8) << small << endl;
  cout << setprecision(10) << small << endl;
  cout << setprecision(12) << small << endl;

  return 0;
}
```



```
1.2e+03
1234
1234.14
1234.1416
1234.141602
1234.14160156
```

Thanks !!