**Standard output.**

(cout)

On most program environments, the standard output by default is the screen, and the C++ stream object defined to access it is *cout*.

For formatted output operations, *cout* is used together with the *insertion operator*, which is written as << (i.e., two "less than" signs).

|  |  |  |
| --- | --- | --- |
| 1 2 3 | cout << "Output sentence"; // prints Output sentence on screen  cout << 8; // prints number 8 on screen  cout << x; // prints the value of x on screen |  |

The *<<* operator inserts the data that follows it into the stream that precedes it. In the examples above, it inserted the literal string Output sentence, the number 8, and the value of variable *x* into the standard output stream cout. Notice that the sentence in the first statement is enclosed in double quotes (") because it is a string literal, while in the last one, *x* is not. The double quoting is what makes the difference; when the text is enclosed between them, the text is printed literally; when they are not, the text is interpreted as the identifier of a variable, and its value is printed instead. For example, these two sentences have very different results:

|  |  |  |
| --- | --- | --- |
| 1 2 | cout << "Hello"; // prints Hello  cout << Hello; // prints the content of variable Hello |  |

Multiple insertion operations (<<) may be chained in a single statement:

|  |  |  |
| --- | --- | --- |
|  | cout << "This " << " is a " << "single C++ statement"; |  |

This last statement would print the text “*This is a single C++ statement”*. Chaining insertions is especially useful to mix literals and variables in a single statement:

|  |  |  |
| --- | --- | --- |
|  | cout << "I am " << age << " years old and my zipcode is " << zipcode; |  |

Assuming the *age* variable contains the value 24 and the *zipcode* variable contains 90064, the output of the previous statement would be:

*I am 24 years old and my zipcode is 90064*

What *cout* does not do automatically is add line breaks at the end, unless instructed to do so. For example, take the following two statements inserting into *cout*:

*cout << "This is a sentence.";  
cout << "This is another sentence.";*

The output would be in a single line, without any line breaks in between. Something like:

*This is a sentence.This is another sentence.*

To insert a line break, a new-line character shall be inserted at the exact position the line should be broken. In C++, a new-line character can be specified as \n (i.e., a backslash character followed by a lowercase n). For example:

|  |  |  |
| --- | --- | --- |
| 1 2 | cout << "First sentence.\n";  cout << "Second sentence.\nThird sentence."; |  |

This produces the following output:

*First sentence.  
Second sentence.  
Third sentence.*

Alternatively, the *endl* manipulator can also be used to break lines. For example:

|  |  |  |
| --- | --- | --- |
| 1 2 | cout << "First sentence." << endl;  cout << "Second sentence." << endl; |  |

This would print:

*First sentence.  
Second sentence.*

The *endl* manipulator produces a newline character, exactly as the insertion of '\n' does; but it also has an additional behavior: the stream's buffer (if any) is flushed, which means that the output is requested to be physically written to the device, if it wasn't already. This affects mainly *fully buffered* streams, and cout is (generally) not a *fully buffered* stream. Still, it is generally a good idea to use *endl* only when flushing the stream would be a feature and '\n' when it would not. Bear in mind that a flushing operation incurs a certain overhead, and on some devices it may produce a delay.

*endl* manipulator doesn’t occupy any memory, whereas \n character occupies 1-byte memory as it is a character.