Printing Blank Lines with println

Document number: **P3142R0**
Date: 2024-02-12
Audience: Library Evolution Working Group
Reply to: **Alan Talbot**
cpp@alantalbot.com

Abstract and Tony Table

This proposal is a pure addition to the Standard Library `println` feature to allow a simpler way to generate a blank line.

C++23	Proposed
`println("");`	`println();`

Motivation

The `print` and `println` functions brought a very welcome simplicity and elegance to text output in C++23. They have changed the standard "Hello, World!" program forever, and very much for the better. However, there is one small missing piece that I think completes the design, namely a way to express a blank line without redundant syntax.

The first objection that comes to mind is: why bother, it's only two more characters. I believe there are both philosophical and practical reasons why this has more impact than extra typing.

Philosophically, it goes against my sense of clean code to be required to pass a meaningless string literal to a function to elicit its default behavior. In fact, the first time I used `println`, I quite naturally tried to write `println()` to get a blank line and was a bit surprised to get a compile error instead.

The practical argument is perhaps more compelling. In my opinion it is a very desirable coding practice to represent small and commonly used character strings with identifiers. This leads to code like:

```
println(BLANK);
```

Which will inevitably result in someone introducing a new (redundant) identifier like:

```
println(SKIP_A_LINE);
```

The point here is that a blank line is a very common use case for which I believe we should provide a standard spelling. If there's a standard way to do it, I for one will put that in my style guide.

The second objection is likely to be, why have this line of code at all? Why not just put a `\n` in the previous line? The answer is enhanced readability and maintainability, and I think that an example is the best way to illustrate this. Turn the page and look at the code for one second, then turn back and answer the question, is there any extra vertical space introduced, and if so where?

```
println("3 large\t\tbananas");
println("2 large\t\teggs\n");
println("280 g (2 c)\tflour");
println("150 g (3/4 c)\tsugar");
println("1 tsp\t\ttable salt (or 2 tsp DCK salt)");
println("1 tsp\t\tbaking soda");
println("1/4 tsp\t\tnutmeg\n");
println("65 g (2/3 c)\twalnut pieces");
println("100 g (2/3 c)\tdark chocolate chips");
```

Not only is it hard to read in the first place, maintaining code like this is fussier and more error-prone because the visible structure of the source code does not match the output—you can't just cut and paste a line of code where you want a blank line. (A similar argument could, and probably should, be made regarding the tabs, but getting the code tabulation to match the output tabulation, at least in the IDE I use, does not work too well.)

Proposed Wording

31.7.2 Header <ostream> synopsis [ostream.syn]

// 31.7.6.3.5, print functions

```
template<class... Args>
   void print(ostream& os, format_string<Args...> fmt, Args&&... args);
template<class... Args>
   void println(ostream& os, format_string<Args...> fmt, Args&&... args);
void println(ostream& os);
```

31.7.4 Header <print> synopsis [print.syn]

// 31.7.10, print functions

…

```
template<class... Args>
   void println(format_string<Args...> fmt, Args&&... args);
void println();
template<class... Args>
   void println(FILE* stream, format_string<Args...> fmt, Args&&... args);
void println(FILE* stream);
```

31.7.6.3.5 Print [ostream.formatted.print]

…

```
template<class... Args>
   void println(ostream& os, format_string<Args...> fmt, Args&&... args);
```

2 Effects: Equivalent to:

```
print(os, "{}\n", format(fmt, std::forward<Args>(args)...));
```

```
void println(ostream& os);
```

Effects: Equivalent to:

```
print(os, "\n");
```

31.7.10 Print functions [print.fun]

…

```
template<class... Args>
   void println(format_string<Args...> fmt, Args&&... args);
```

3 Effects: Equivalent to:

```
println(stdout, fmt, std::forward<Args>(args)...);
```

```
void println();
```

Effects: Equivalent to:

```
println(stdout);
```

```
template<class... Args>
   void println(FILE* stream, format_string<Args...> fmt, Args&&... args);
```

4 Effects: Equivalent to:

```
print(stream, "{}\n", format(fmt, std::forward<Args>(args)...));
```

```
void println(FILE* stream);
```

Effects: Equivalent to:

```
print(stream, "\n");
```

Postscript

Preheat the oven to 350° F (175° C). Spray or grease an 8” square cake pan. (A 9” loaf pan may also be used, but a cake pan gives a more even baking in less time.)

Put the bananas in a stand mixer and run until they are well mashed. Add the eggs and run until the eggs are well beaten.

Combine the flour, sugar, salt, baking soda and nutmeg in a large mixing bowl. Whisk well, then add the dry mixture to the stand mixer and stir until combined.

Add the walnuts and chocolate chips and stir by hand.

Pour the batter into the prepared pan and bake for 45 minutes with convection to an internal temperature of 200° F (93° C). Cool on a rack.

(Adapted from The Fannie Farmer Cookbook, 1979 edition)