1.6 — Whitespace and basic formatting

BY ALEX ON JUNE 1ST, 2007 | LAST MODIFIED BY ALEX ON FEBRUARY 28TH, 2018

Whitespace is a term that refers to characters that are used for formatting purposes. In C++, this refers primarily to spaces, tabs, and (sometimes) newlines. The C++ compiler generally ignores whitespace, with a few minor exceptions.

Consequently, the following statements all do the exact same thing:

```
std::cout << "Hello world!";

std::cout               <<            "Hello world!";

        std::cout <<        "Hello world!";

std::cout
    << "Hello world!";
```

Even the last statement with the newline in it compiles just fine.

The following functions all do the same thing:

```
int add(int x, int y) { return x + y; }

int add(int x, int y) {
    return x + y; }

int add(int x, int y)
{    return x + y; }

int add(int x, int y)
{
    return x + y;
}
```

One exception where the C++ compiler *does* pay attention to whitespace is inside quoted text, such as `"Hello world!"`.

```
"Hello world!"
```

is different than

```
"Hello     world!"
```

and each prints out exactly as you'd expect.

Newlines are not allowed in quoted text:

```
std::cout << "Hello
     world!" << std::endl; // Not allowed!
```

Another exception where the C++ compiler pays attention to whitespace is with // comments. Single-line comments only last to the end of the line. Thus doing something like this will get you in trouble:

```
std::cout << "Hello world!" << std::endl; // Here is a single-line comment
this is not part of the comment
```

Basic formatting

Unlike some other languages, C++ does not enforce any kind of formatting restrictions on the programmer (remember, trust the programmer!). Many different methods of formatting C++ programs have been developed throughout the years, and you will find disagreement on which ones are best. Our basic rule of thumb is that the best styles are the ones that produce the most readable code, and provide the most consistency.

Here are our recommendations for basic formatting:

1) Your tabs should be set to 4 spaces (most IDEs have a setting where you can configure this). Some IDEs default to 3 spaces, which is a little unconventional but okay.

You can use tabs or spaces for indentation. Most major companies and style guides recommend using spaces (either 2 or 4 at a time) because it produces more consistency across all editors. Personally, I don't find this reason compelling, and prefer tabs because they are easier to use.

(The tabs vs spaces is a debate that has gone on for 50+ years now -- there's no right answer. Most important thing is to be consistent with the code around yours and the standards of your company, if applicable).

2) The braces that tell where a function begins and ends should be aligned with the function name, and be on their own lines:

```
int main()
{
}
```

Although some coders prefer other styles, this one is the most readable and least error prone since your brace pairs should always be indented at the same level. If you get a compiler error due to a brace mismatch, it's very easy to see where.

3) Each statement within braces should start one tab (or 4 spaces) in from the opening brace of the function it belongs to. For example:

```
#include <iostream>
int main()
{
    std::cout << "Hello world!" << std::endl; // tabbed in one tab (4 spaces)
    std::cout << "Nice to meet you." << std::endl; // tabbed in one tab (4 spaces)
}
```

4) Lines should not be too long. Typically, 72, 78, or 80 characters is the maximum length a line should be. If a line is going to be longer, it should be broken (at a reasonable spot) into multiple lines. This can be done by indenting each subsequent line with an extra tab, or if the lines are similar, by aligning it with the line above (whichever is easier to read).

```
#include <iostream>
int main()
{
    std::cout << "This is a really, really, really, really, really, really, really, " <<
        "really long line" << std::endl; // one extra indentation for continuation line

    std::cout << "This is another really, really, really, really, really, really, really, " <<
                 "really long line" << std::endl; // text aligned with the previous line for continuati
on line

    std::cout << "This one is short" << std::endl;
}
```

5) If a long line that is broken into pieces is broken with an operator (eg. << or +), the operator should be placed at the end of the line, not the start of the next line:

```
    std::cout << "This is a really, really, really, really, really, really, really, " <<
        "really long line" << std::endl;
```

Not

```
    std::cout << "This is a really, really, really, really, really, really, really, "
        << "really long line" << std::endl;
```

This makes it more obvious from looking at the first line that the next line is going to be a continuation.

6) Use whitespace to make your code easier to read.

Harder to read:

```
nCost = 57;
nPricePerItem = 24;
```

```
nValue = 5;
nNumberOfItems = 17;
```

Easier to read:

```
nCost          = 57;
nPricePerItem  = 24;
nValue         = 5;
nNumberOfItems = 17;
```

Harder to read:

```
std::cout << "Hello world!" << std::endl; // std::cout and std::endl live in the iostream library
std::cout << "It is very nice to meet you!" << std::endl; // these comments make the code hard to read
std::cout << "Yeah!" << std::endl; // especially when lines are different lengths
```

Easier to read:

```
std::cout << "Hello world!" << std::endl;                  // std::cout and std::endl live in the iostre
am library
std::cout << "It is very nice to meet you!" << std::endl;  // these comments are easier to read
std::cout << "Yeah!" << std::endl;                         // especially when all lined up
```

Harder to read:

```
// std::cout and std::endl live in the iostream library
std::cout << "Hello world!" << std::endl;
// these comments make the code hard to read
std::cout << "It is very nice to meet you!" << std::endl;
// especially when all bunched together
std::cout << "Yeah!" << std::endl;
```

Easier to read:

```
// std::cout and std::endl live in the iostream library
std::cout << "Hello world!" << std::endl;

// these comments are easier to read
std::cout << "It is very nice to meet you!" << std::endl;

// when separated by whitespace
std::cout << "Yeah!" << std::endl;
```

We will follow these conventions throughout this tutorial, and they will become second nature to you. As we introduce new topics to you, we will introduce new style recommendations to go with those features.

Ultimately, C++ gives you the power to choose whichever style you are most comfortable with, or think is best. However, we highly recommend you utilize the same style that we use for our examples. It has been battle tested by thousands of programmers over billions of lines of code, and is optimized for success.

1.7 -- Forward declarations and definitions

Index

1.5 -- A first look at operators

Share this: