On the virtues of the trailing comma

devblogs.microsoft.com/oldnewthing/20240209-00

February 9, 2024

Raymond Chen

Many programming languages allow trailing commas in lists.

C, C++, C# (and probably other languages) permit a trailing comma after the last enumerator:

```
enum Color
{
    Red,
    Blue,
    Green,
    //   ^ trailing comma
};
```

They also allow a trailing comma in list initializers.

```
// C, C++
Thing a[] = {
    { 1, 2 },
    { 3, 4 },
    { 5, 6 },
    //      ^ trailing comma
};

// C#
Thing[] a = new[] {
    new Thing {
        Name = "Bob",
        Id = 31415,
        //        ^ trailing comma
    },
    new Thing {
        Name = "Alice",
        Id = 2718,
        //       ^ trailing comma
    },
//   ^ trailing comma
};

Dictionary d = new Dictionary<string, Thing>() {
    ["Bob"] = new Thing("Bob") { Id = 31415 },
    ["Alice"] = new Thing("Alice", 2718),
    //                                  ^ trailing comma
};
```

These trailing commas are convenient when you arrange for each element to appear on its own line, like we did in the examples above. It lets you rearrange the items by moving lines around without having to worry about having to add a comma to an element when it moves out of the final position, or removing a comma from the element that moved into the final position.

It also reduces merge risk when people modify the list. For example, if somebody adds a new color “Black” to the end, they won’t have to touch any of the other lines, which means that a change from “Blue” to “LightBlue” won’t result in a merge conflict.

And even when there is a merge conflict due to two simultaneous adds, you can easily resolve it by accepting both.

```
enum Color
{
    Red,
    Blue,
    Green,
<<< VERSION 1
    Black,
|||
    White,
<<< VERSION 2
};
```

To resolve this, you can just delete all the conflict markers.

```
enum Color
{
    Red,
    Blue,
    Green,
    Black,
    White,
};
```

If your code didn't use trailing commas, the merge would be messier:

```
enum Color
{
    Red,
    Blue,
<<< VERSION 1
    Green,
    Black
|||
    Green,
    White
<<< VERSION 2
};
```

And if you have a lot of these merges to deal with, you might forget to insert a comma after "Black":

```
enum Color
{
    Red,
    Blue,
    Green,
    Black // ⇐ oops, forgot a comma
    White
};
```

Since the trailing comma reduces the number of lines of code that have to be modified when the list is extended, it also makes `git blame` more accurate. Without the trailing comma, a `git blame` on `enum Color` would blame the person who added “Black” for also being the last person to modify the “Green” line. If you’re investigating a problem with “Green”, you might ask that person for help, and they’ll say, “Oh no, I didn’t add ‘Green’. I added ‘Black’. You’ll have to dig further back into the history to figure out who added ‘Green’.”

C	C++	C#
Thank	you	for

Java	JSON	JavaScript
supporting	Not you	trailing

Rust	Go	Python
commas	in	lists

Bonus chatter: The trailing comma also makes it easier for code generators, since they can just emit a comma after each element and not have to worry about suppressing the final comma.

Bonus bonus chatter: But why not go all the way and allow a trailing comma in parameter lists?

```
SomeFunction(1, 2, );
//              ^ trailing comma not allowed
```

I suspect the primary reason is “nobody asked for it.” Variadic functions are relatively uncommon, so this is not something that code generators stumble across. Also, that extra comma just plain looks weird.

Overloaded functions could pose a parsing problem. If there are 2-parameter and 3-parameter overloads of `SomeFunction`, is this a call to the two-parameter overload, or is it a call to the three-parameter overload with some sort of default?

Bonus bonus bonus chatter: JavaScript, Rust, and Ruby allow a trailing comma in parameter lists.

Bonus bonus bonus bonus chatter: In the Pascal programming language, the semicolon is a statement separator, not a statement terminator, so you can write

```
begin
  i := 1;
  j := 2  (* no trailing semicolon *)
end
```

In practice, everybody puts a semicolon just before the end. Imaging rearranging two lines of code and having to adjust semicolons.