What does 1#J mean? A strange corner case of the printing of special values

devblogs.microsoft.com/oldnewthing/20130228-01

February 28, 2013

Raymond Chen

As a puzzle, commenter nobugz asks, "What kind of infinity is `1.#J`?"

```
double z = 0;
printf("%.2f", 1/z);
```

Now, the division by zero results in IEEE positive infinity, would would normally be printed as `1#INF`. But the catch here is that the print format says "Display at most two places after the decimal point." But where is the decimal point in infinity?

The Visual C runtime library arbitrarily decided that all of the exceptional values have one digit before the decimal (namely, the `"1"`). Actually, it turns out that this puzzle might be an answer to Random832's question, "What's the 1 for?" Maybe the 1 is there so that there is a digit at all.

Okay, so now you have one digit before the decimal (the `"1"`), and now you need to show at most two places after the decimal. But `"#INF"` is too long to fit into two characters. The C runtime then says, "Well, then I'd better round it off to two places, then."

The first character is `"#"`. The second character is `"I"`. Now we need to round it. That's done by inspecting the third character, which is `"N"`. We all learned in grade school that you round up if the next digit is 5 or greater. And it so happens that the code point for `"N"` is numerically higher than the code point for `"5"`, so the value is rounded up by incrementing the previous digit. Incrementing `"I"` gives you `"J"`.

That's why printing IEEE positive infinity to two places gives you the strange-looking `"1#J"`. The `J` is an `I` that got rounded up.

I doubt this behavior was intended; it's just a consequence of taking a rounding algorithm intended for digits and applying it to non-digits.

Of course, in phonetics, rounding an *i* produces a *ü*. Imagine the nerdiness of rounding `"1#INF"` to two places and producing `"1#Ü"`. That would have been awesome.

Raymond Chen

Follow