The C++ preprocessor doesn't understand anything about C++, and certainly not templates

devblogs.microsoft.com/oldnewthing/20200508-52

May 8, 2020

Raymond Chen

The C and C++ preprocessors are like the Windows command prompt batch language: They implement a very simple language that people still use for some reason, usually by pushing it way beyond its original design boundaries.

The preprocessors don't really understand the C or C++ language. It does, fortunately, use the same arithmetic operators that the C and C++ language uses, but its understanding of them is limited to integers.

The treatment of commas in preprocessor arguments is very simple: Commas separate macro arguments. The only way to protect a comma is to enclose it in matched parentheses.

```
#define M(cond, action) if (cond) action // horrible macro

M(function(2, 3), return) // okay
```

Note that the less-than and greater-than signs do not protect commas. This can create confusion if you want to pass a template instantiation as a macro parameter.

```
M(std::is_same_v<T, U>, return); // doesn't compile
```

The preprocessor isn't smart enough to realize that what you passed was intended to be a template instantiation. After all, maybe you wanted this:

```
M(value < 0 || height > 1000, return out_of_range);
```

In this case, the less-than and greater-than signs are intended to be the comparison operators. But from the preprocessor's point of view, the angle brackets in

```
    std::is_same_v< T, U >
```

and

```
    value < 0 || height > 1000
```

are basically the same thing. It has no way of knowing that the first case of matched angle brackets is a template instantiation, whereas the second is just a pair of comparison operators. Not even the compiler knows, because we are still preprocessing. Compilation hasn't happened yet.

The solution is to insert seemingly-spurious parentheses around the macro argument.

```
M((std::is_same_v<T, U>), return);
```

Raymond Chen

Follow