What is Pragma in C?

In C, pragma is a directive to turn features on or off. That is to say, it gives instructions to the compiler directly. The difference between pragma and a lot of other macros is that it is compiler dependent.

```
#pragma directive_name
```

Use Cases

There are many different use cases for it, for example, one of our most common usages is `#pragma once`

> So here we can note the difference between `pragma` and a common macro such as `ifndef`, where pragma uses the once directive to indicate anything in the header to be only included once. `ifndef` is merely a conditional control for the program.

And while there are many other common uses such as `#pragma startup` and `#pragma exit` which when used with a `function_name`, allows the program to run a function, that is, as a preprocessor, means that it can run even before the program has entered the main function! Or `#pragma warn` to set much finer detail to showing or hiding warnings from the compiler.

But, here we can see a much cooler use case, that is `#pragma poison`, which when given an identifier which for example, may be a function. The function could be written or included from the C standard library of function implementations. But here is something a bit more unique,

```
#define STRINGIFY(x) #x

#ifndef MY_FEATURE
  #define CHECK_MY_FEATURE _Pragma(STRINGIFY(message("Warning: " MY_FEATURE " is
not defined; using fallback behavior")))
#else
  #define CHECK_MY_FEATURE
#endif

// Invoke the macro at global scope or within a function.
CHECK_MY_FEATURE
```

Here, it checks for `MY_FEATURE`, which if not defined, will emit a compile-time message, warning the user. Otherwise, it does nothing. Overall, it shows how versatile pragma is, even though it is not completely cross-compatible between compilers, but within the context of modern C and its preprocessor methods, it is a much needed tool for many use cases.