7 Pragmas

The `#pragma' directive is the method specified by the C standard for providing additional information to the compiler, beyond what is conveyed in the language itself. Three forms of this directive (commonly known as *pragmas*) are specified by the 1999 C standard. A C compiler is free to attach any meaning it likes to other pragmas.

GCC has historically preferred to use extensions to the syntax of the language, such as `__attribute__`, for this purpose. However, GCC does define a few pragmas of its own. These mostly have effects on the entire translation unit or source file.

In GCC version 3, all GNU-defined, supported pragmas have been given a `GCC` prefix. This is in line with the `STDC` prefix on all pragmas defined by C99. For backward compatibility, pragmas which were recognized by previous versions are still recognized without the `GCC` prefix, but that usage is deprecated. Some older pragmas are deprecated in their entirety. They are not recognized with the `GCC` prefix. See Obsolete Features.

C99 introduces the `_Pragma` operator. This feature addresses a major problem with `#pragma': being a directive, it cannot be produced as the result of macro expansion. `_Pragma` is an operator, much like `sizeof` or `defined`, and can be embedded in a macro.

Its syntax is `_Pragma (`*string-literal*`)`, where *string-literal* can be either a normal or wide-character string literal. It is destringized, by replacing all `\\' with a single `\' and all `\"' with a `"'. The result is then processed as if it had appeared as the right hand side of a `#pragma' directive. For example,

```
_Pragma ("GCC dependency \"parse.y\"")
```

has the same effect as `#pragma GCC dependency "parse.y"`. The same effect could be achieved using macros, for example

```
#define DO_PRAGMA(x) _Pragma (#x)
DO_PRAGMA (GCC dependency "parse.y")
```

The standard is unclear on where a `_Pragma` operator can appear. The preprocessor does not accept it within a preprocessing conditional directive like `#if'. To be safe, you are probably best keeping it out of directives other than `#define', and putting it on a line of its own.

This manual documents the pragmas which are meaningful to the preprocessor itself. Other pragmas are meaningful to the C or C++ compilers. They are documented in the GCC manual.

GCC plugins may provide their own pragmas.

`#pragma GCC dependency`

`#pragma GCC dependency` allows you to check the relative dates of the current file and another file. If the other file is more recent than the current file, a warning is issued. This is useful if the current file is derived from the other file, and should be

regenerated. The other file is searched for using the normal include search path. Optional trailing text can be used to give more information in the warning message.

```
#pragma GCC dependency "parse.y"
#pragma GCC dependency "/usr/include/time.h" rerun
fixincludes
```

`#pragma GCC poison`

Sometimes, there is an identifier that you want to remove completely from your program, and make sure that it never creeps back in. To enforce this, you can *poison* the identifier with this pragma. `#pragma GCC poison` is followed by a list of identifiers to poison. If any of those identifiers appears anywhere in the source after the directive, it is a hard error. For example,

```
#pragma GCC poison printf sprintf fprintf
sprintf(some_string, "hello");
```

will produce an error.

If a poisoned identifier appears as part of the expansion of a macro which was defined before the identifier was poisoned, it will *not* cause an error. This lets you poison an identifier without worrying about system headers defining macros that use it.

For example,

```
#define strrchr rindex
#pragma GCC poison rindex
strrchr(some_string, 'h');
```

will not produce an error.

`#pragma GCC system_header`

This pragma takes no arguments. It causes the rest of the code in the current file to be treated as if it came from a system header. See System Headers.

`#pragma GCC warning`
`#pragma GCC error`

`#pragma GCC warning "message"` causes the preprocessor to issue a warning diagnostic with the text `message'. The message contained in the pragma must be a single string literal. Similarly, `#pragma GCC error "message"` issues an error message. Unlike the `#warning' and `#error' directives, these pragmas can be embedded in preprocessor macros using `_Pragma'.