Implementation defined behavior control

Implementation defined behavior is controlled by `#pragma` directive.

Syntax

#pragma *pragma-params*	(1)	
_Pragma(*string-literal* **)**	(2)	(since C++11)

1) Behaves in implementation-defined manner.

2) Removes the L prefix (if any), the outer quotes, and leading/trailing whitespace from *string-literal*, replaces each `\"` with `"` and each `\\` with `\`, then tokenizes the result (as in translation phase 3), and then uses the result as if the input to `#pragma` in (1).

Explanation

Pragma directive controls implementation-specific behavior of the compiler, such as disabling compiler warnings or changing alignment requirements. Any pragma that is not recognized is ignored.

Non-standard pragmas

The ISO C++ language standard does not require the compilers to support any pragmas. However, several non-standard pragmas are supported by multiple implementations:

#pragma STDC

ISO C language standard requires that C compilers support the following three pragmas, and some C++ compiler vendors support them, to varying degrees, in their C++ frontends:

#pragma STDC FENV_ACCESS *arg*	(1)
#pragma STDC FP_CONTRACT *arg*	(2)
#pragma STDC CX_LIMITED_RANGE *arg*	(3)

where *arg* is either **ON**, **OFF**, or **DEFAULT**.

1) If set to **ON**, informs the compiler that the program will access or modify floating-point environment, which means that optimizations that could subvert flag tests and mode changes (e.g., global common subexpression elimination, code motion, and constant folding) are prohibited. The default value is implementation-defined, usually **OFF**.

2) Allows *contracting* of floating-point expressions, that is optimizations that omit rounding errors and floating-point exceptions that would be observed if the expression was evaluated exactly as written. For example, allows the implementation of `(x * y) + z` with a single fused multiply-add CPU instruction. The default value is implementation-defined, usually **ON**.

3) Informs the compiler that multiplication, division, and absolute value of complex numbers may use simplified mathematical formulas $(x+iy)\times(u+iv) = (xu-yv)+i(yu+xv)$, $(x+iy)/(u+iv) = [(xu+yv)+i(yu-xv)]/(u^2+v^2)$, and $|x+iy| = \sqrt{x^2+y^2}$, despite the possibility of intermediate overflow. In other words, the programmer guarantees that the range of the values that will be passed to those function is limited. The default value is **OFF**.

The behavior of the program is undefined if any of the three pragmas above appear in any context other than outside all external declarations or preceding all explicit declarations and statements inside a compound statement.

Note: compilers that do not support these pragmas may provide equivalent compile-time options, such as gcc's `-fcx-limited-range` and `-ffp-contract`.

#pragma once

`#pragma once` is a non-standard pragma that is supported by the vast majority of modern compilers . If it appears in a header file, it indicates that it is only to be parsed once, even if it is (directly or indirectly) included multiple times in the same source file.

Standard approach to preventing multiple inclusion of the same header is by using include guards :

```
#ifndef LIBRARY_FILENAME_H
#define LIBRARY_FILENAME_H
// contents of the header
#endif /* LIBRARY_FILENAME_H */
```

So that all but the first inclusion of the header in any translation unit are excluded from compilation. All modern compilers record the fact that a header file uses an include guard and do not re-parse the file if it is encountered again, as long as the guard is still defined (see e.g. gcc (https://gcc.gnu.org/onlinedocs/cpp/Once-Only-Headers.html)).

With `#pragma once`, the same header appears as

```
#pragma once
```

```
// contents of the header
```

Unlike header guards, this pragma makes it impossible to erroneously use the same macro name in more than one file. On the other hand, since with `#pragma once` files are excluded based on their filesystem-level identity, this can't protect against including a header twice if it exists in more than one location in a project.

`#pragma pack`

This family of pragmas control the maximum alignment for subsequently defined class and union members.

`#pragma pack(`*`arg`*`)`	(1)
`#pragma pack()`	(2)
`#pragma pack(push)`	(3)
`#pragma pack(push, `*`arg`*`)`	(4)
`#pragma pack(pop)`	(5)

where *arg* is a small power of two and specifies the new alignment in bytes.

1) Sets the current alignment to value *arg*.
2) Sets the current alignment to the default value (specified by a command-line option).
3) Pushes the value of the current alignment on an internal stack.
4) Pushes the value of the current alignment on the internal stack and then sets the current alignment to value *arg*.
5) Pops the top entry from the internal stack and then sets (restores) the current alignment to that value.

`#pragma pack` may decrease the alignment of a class, however, it cannot make a class overaligned.

See also specific details for GCC (https://gcc.gnu.org/onlinedocs/gcc/Structure-Layout-Pragmas.html) and MSVC (https://docs.microsoft.com/en-us/cpp/preprocessor/pack) .

This section is incomplete
Reason: Explain the effects of this pragmas on data members and also the pros and cons of using them. Sources for reference:

- Stack Overflow (https://stackoverflow.com/questions/3318410/pragma-pack-effect)

This section is incomplete
Reason: no example

References

- C++23 standard (ISO/IEC 14882:2023):
  - 15.9 Pragma directive [cpp.pragma]
- C++20 standard (ISO/IEC 14882:2020):
  - 15.9 Pragma directive [cpp.pragma]
- C++17 standard (ISO/IEC 14882:2017):
  - 19.6 Pragma directive [cpp.pragma]
- C++14 standard (ISO/IEC 14882:2014):
  - 16.6 Pragma directive [cpp.pragma]
- C++11 standard (ISO/IEC 14882:2011):
  - 16.6 Pragma directive [cpp.pragma]
- C++98 standard (ISO/IEC 14882:1998):
  - 16.6 Pragma directive [cpp.pragma]

See also

C documentation for **Implementation defined behavior control**

External links

1. C++ pragmas in Visual Studio (https://docs.microsoft.com/en-us/cpp/preprocessor/pragma-directives-and-the-pragma-keyword)
2. Pragmas (https://gcc.gnu.org/onlinedocs/gcc/Pragmas.html) accepted by GCC
3. Individual pragma descriptions (https://www.ibm.com/support/knowledgecenter/en/SSGH3R_16.1.0/com.ibm.xlcpp161.aix.doc/compiler_ref/pragma_descriptions.html) and Standard pragmas

(https://www.ibm.com/support/knowledgecenter/en/SSGH3R_16.1.0/com.ibm.xlcpp161.aix.doc/language_ref/std_pragmas.html) in IBM AIX XL C 16.1

4. Appendix B. Pragmas (https://download.oracle.com/docs/cd/E19422-01/819-3690/Pragmas_App.html#73499) in Sun Studio 11 C++ User's Guide
5. Intel C++ compiler pragmas (https://software.intel.com/content/www/us/en/develop/documentation/cpp-compiler-developer-guide-and-reference/top/compiler-reference/pragmas.html)
6. Release nodes (includes pragmas) (https://h20565.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=4268164&docLocale=en_US&docId=emr_na-c02653979) for HP aCC A.06.25