Trimming whitespaces before line splicing

Document #: P2223R2
Date: 2021-04-13
Project: Programming Language C++
Audience: EWG, CWG
Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Abstract

We propose to make trailing whitespaces after \ non-significant.

This paper has been extracted from P2178R0 [1] and is part of a larger effort from SG-16 to improve lexing.

Revisions

Revision 2

- Wording changes as per CWG review (April 14).

Revision 1

- Make it clearer that raw strings are not impacted.
- Remove CWG 1698 fix following EWG guidance.

Making trailing whitespaces non-significant

There is a divergence of implementation in how compilers handle spaces between a backslash and the end of a line.

```
int main() {
    int i = 1
    //  \
    + 42
    ;
    return i;
}
```

EDG(tested with icc front-end) GCC and Clang will trim the whitespaces after the backslash - and return `1` - MSVC will not and return `43`. Both strategies are valid as part of phase 1 ”implementation-defined mapping”. There is a clang issue about this #15509

To avoid this surprising implementation divergence we proposed that an implementation must trim all trailing whitespaces before handling \ splicing. This is reinforced by the fact that IDES and tools (code formatter for example) may discard such whitespaces. The Google-style guidelines forbid trailing whitespaces.

An additional or alternative approach is to deprecate \ that are not part of a preprocessor directive. We are not proposing this at this time.

This proposal may be a silent breaking change for code that is only compiled with MSVC. A quick analysis of the packages available in VCPKG didn’t find any trailing whitespaces after backslashes.

We have not been able to measure the impact of this proposed change in the MSVC ecosystem. Other compilers, and all code supported by these compilers would be unaffected. However, neither tools nor people can be expected to reliably preserve invisible character. The lack of usefulness and the brittleness of relying on this behavior makes it unlikely to be relied upon.

GCC, Clang, and ICC provide a warning enabled by default **warning: backslash and newline separated by space**.

Example

An example of actual issue encountered in the while is people using ASCII art and diagram in comments (Thanks JF for the museum-worthy art piece).

```
int main() {
    int hax = 0;
    //  _    _
    // | |  | |
    // | |__| | __ ___  __
    // |  __  |/ _` \ \/ /
    // | |  | | (_| |>  <
    // |_|  |_|\__,_/_/\_\
    hax = 1337;
    return hax;
}
```

String literals

Another interesting example

```
auto str  "\<space>
";
```

Is an empty string in GCC, ICC, Clang and the string `"\ "` in MSVC. As `'\ '` is not a valid escape sequence, the above code is ill-formed in MSVC.

Raw string literals

The proposed trimming is intended to be reversed in raw string literals. However, the reversal of line splicing in raw strings might not be sufficiently specified, leading to some implementation divergence and open GCC issues #91412, #43606, which the present paper doesn't intend to address.

The proposed wording modifies splicing and splicing is entirely reversed in raw string literals (5.4.3.1), as such we the proposed wording does not change the current raw string literals behavior.

Whitespaces?

The proposed wording as well as the standard don't specify what characters are considered whitespaces. This will be addressed separately (see P2178R0 [1]), but the intent is that characters considered whitespaces for the purpose of lines splicing be the same as whitespace characters in the rest of the grammar. This includes tabs and other characters considered whitespaces by the Unicode standard.

C compatibility

In effect, the "implementation defined mapping: in phase one makes the content of the program implementation defined, and it is a valid behavior in C and C++ alike to trimm or not trailing whitespaces. The proposed change only requires that all C++ compilers do this trimming, and therefore doesn't affect C compatibility.

Proposed Wording

Modify 5.2.1.2 of [N4861] as follows:

� Lexical conventions [lex]

� Phases of translation [lex.phases]

Each ~~instance~~ sequence of a backslash character (\) immediately followed by zero or more whitespace characters other than new-line followed by a new-line character is deleted, splicing physical source lines to form logical source lines. Only the last backslash on any physical source line shall be eligible for being part of such a splice. Except for splices reverted in a raw string literal, if a splice results in a character sequence that matches the syntax of a *universal-character-name*, the behavior is undefined. A source file that is not empty and that does not end in a new-line character, or that ends in a ~~new-line character immediately preceded by a backslash character before any such splicing takes place~~ splice, shall be processed as if an additional new-line character were appended to the file.

References

[1] Corentin Jabot. P2178R0: Misc lexing and string handling improvements. `https://wg21.link/p2178r0`, 6 2020.

[UAX-14] *UNICODE LINE BREAKING ALGORITHM*
`https://www.unicode.org/reports/tr14/`

[Unicode] Unicode 13
`http://www.unicode.org/versions/Unicode13.0.0/`

[N4861] Richard Smith *Working Draft, Standard for Programming Language C++*
`https://wg21.link/N4861`