C and C++ Compatibility Study Group Meeting Minutes (Aug 2022)

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N3052
SG Meeting Date: 2022-08-15

Mon Aug 15, 2022 at 11:00am EST

Attendees

Aaron Ballman	WG21/WG14	chair, scribe
Corentin Jabot	WG21/(14)	
Hubert Tong	WG21/(14)	
Tom Honermann	WG21	
Christof Merrwald	WG21	
Steve Downey	WG21	

Insufficient folks for quorum, all discussions are informal.

P2620R1: Lifting artificial restrictions on universal character names

CJ: proposes to lift a restriction in C++ on use of UCNs in an identifier. The restriction feels artifical. C disallows them in string literals, but that surprises users in practice. This tries to lift the restriction in C++ and in C we should lift the restriction for identifiers and string literals.

CJ: the restriction was put in place because of concerns around keywords. If you have a UCN and the identifier denotes a keyword, what should happen? This was brought up on the reflectors as a possible concern we should discuss.

HT: strongly against allowing the spelling of effective keywords with UCNs, but the idea of allowing UCNs to spell keywords and be identifiers is intriguing; unsure how I feel about that.

TH: that makes sense. It'd be useful to update the title of the paper to note it's only applicable to identifiers

CJ: can do

CJ: this is a low-priority paper; we don't want to spend too much committee time on it

TH: I'm not sure the wording is right as-is, but I expect the wording to be reasonably minimal and not take up a lot of committee time

TH: the choices for keywords: (1) don't allow UCNs, (2) recognize UCN as being a character and forms a keyword

HT: the wording changes things, but it might be good. We need C committee members to see if they honestly believe in the C99 rationale. The models are different between the committees, but the differences might matter.

HT: do other languages allow UCN-like things in keyword or identifiers?

CJ: not that I know of

TH: most other languages don't need them because they work in Unicode anyway

HT: what happens to a macro name that matches the form of the keyword?

AB: C2x 7.1.2p5 says "the program shall not have any macros with names lexically identical to keywords currently defined prior to the inclusion of the header or when any macro defined in the header is expanded."

TH: so this basically removes a diagnostic for implementations?

CJ: basically, yes

AB: the C++ wording is at http://eel.is/c++draft/library#macro.names-2

HT: recursive formation of UCNs from other UCNs

CJ: do you want me to address that somehow?

HT: it seems to fall out from the paper, but the previous version was more ambiguous

HT: the other thing about the new wording is it might cause changes to tokenization behavior if you have an invalid UCN, it forms the stray backslash again (invalid in terms of not enough characters).

CJ: this is already the case, I believe

HT: I'm more wondering if moving the words into the grammar modifies *when* we determine what is a valid UCN

CJ: I don't *think* it changes when that happens, but maybe there is a better way to express that. The design intent is not to change behavior.

HT: I'm pretty sure the effect is correct, but it'd be nice for the paper to spell that out

HT: I think we can use UCNs for the exponent in a number (the "e" part), but it's not an identifier.

TH: that's more keyword like than identifier like

AB: what about non-directives from the preprocessor?

CJ: the UCN is replaced immediately, shouldn't be an issue probably?

HT: the word "defined" is not a keyword, it's a token with special meaning for the preprocessor, which is another issue.

CJ: I think the choice really boils down to whether we want to allow UCN everywhere or not. Maybe we don't want to invest time in this if it only impacts identifiers (and string literals in C).

HT: I understand why we'd want it, but I don't understand much why we would want it for the basic character set. Even if we don't want to persue the paper at all, we should at least say why so that it's in the record.

AB: my feeling is we should either go fully or just leave it alone for the time being

TH: I would be fine leaving it alone; the code generation motiviation is pretty weak

HT: note that the wording change in terms of where UCNs are recognized might be editorial and worth keeping, maybe? Don't throw the baby out with the bathwater if there are good bits to keep.

AB: I'd also be happy if we got C and C++ more in line in this space in terms of UCNs within string literals.

CJ: I can file core issues for the good bits from this paper, if I drop it; but we do need someone to write the string literal paper for WG14 though. I don't know C's model well enough to feel comfortable doing that alone.

HT: we should definitely document what edge cases we ran into and how we came to the decision to not persue this paper, so people don't have to rediscover this information for themselves. Afterwards, the question is: do we ask Evolution to look at that paper?

HT: the category where you have the unclosed string literal was historically used for multiline string literals in C, so changing that behavior would break code.

HT: string literals with a newline in them are a historical extension for C compilers (to insert the newline in the literal), so we have to be careful about that

AB: depends on how "modern" those implementations are. If they only support C89, they're less compelling to WG14 (generally) because they're not keeping up with the language.

Wrapup

Aaron: Sept meeting is on the 19th (also a Monday at the same time as this meeting) and we'll for sure be covering a paper about operator dot.

End at 12:08pm EST