Proposal for C2x
WG14 N3124

Aligning Universal Character Names Constraints with C++

Reply-to: Corentin Jabot (corentinjabot@gmail.com)
Document No: N3124
Date: 2024-22-04

Problem

Both C and C++ recently added $, @, and ` to the list of characters whose support in source file is mandated.
C++ added it to the basic character set

C instead mandates these characters must be supported, without extending the basic character set. This was done this way because

- C does not allow UCNs to designate basic character elements in string literals.
- "\u0024" sometimes appear in source code

This poses a few issues

- Implementations do not really support UCNs outside of identifiers, a stray \u0040 in a program for example will be diagnosed as an invalid identifier by most implementations, so the C standard here is supporting something that compilers have no good way to support.
- Not allowing elements of the basic character set to be formed by a UCN outside of string literals makes a lot of sense, as doing so would force implementers to support parsing UCNs as punctuators and worse, if we were to allow basic characters elements as UCNs, keywords may contain UCNs. SG22 previously discussed how this would add complexity for preprocessor directive and macros like `defined`, and contextual keywords, and that going down that road would be a significant effort for little benefit. On the flip side, there is no reason to impose any restrictions on string literals
- Some implementations - GCC and compatible implementations, do support $ in identifiers. This makes `int \u0024 = 0;` valid on those implementations in C but not in C++.

- C++ may decide in the future to use $, @ or ` if there is a pressing need for more punctuators. Supporting these as UCNs outside of strings would be an impediment to that.

The best way to fix this divergence between the two languages, is for C to adopt the same rules as C++ by:

- Allowing any UCNs in strings and characters constants
- Not allowing UCNs designating elements of the basic character set anywhere else.

Breaking change?

The proposed wording doesn't exactly allow \u0024 as an extension to support $.
I haven't found a single use of \u0024 outside of identifiers outside of GCC's test suite.

Wording

5.2 Environmental considerations

5.2.1 Character sets

Both the basic source and basic execution character sets shall have the following members:

The 26 uppercase letters of the Latin alphabet
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z

The 26 lowercase letters of the Latin alphabet
a b c d e f g h i j k l m
n o p q r s t u v w x y z

The 10 decimal digits
0 1 2 3 4 5 6 7 8 9

The following ~~29~~ 32 graphic characters
! " # % & ' ( ) * + , - . / :
; < = > ? [\ ] ^ _ { | } ~
@ $ `

the space character, and control characters representing horizontal tab, vertical tab, and form feed. The representation of each member of the source and execution basic character sets shall fit in a byte. In both the source and execution basic character sets, the value of each character after 0 in the above list of decimal digits shall be one greater than the value of the previous. In source files, there shall be some way of indicating the end of each line of text; this document treats such an end-of-line indicator as if it were a single new-line character. In the basic execution character set, there shall be control characters representing alert, backspace, carriage return, and new line. If any other characters are encountered in a source file (except in an identifier, a character constant, a string literal, a header name, a comment, or a preprocessing token that is never converted to a token), the behavior is undefined.
4 A letter is an uppercase letter or a lowercase letter as defined above; in this document the term does not include other characters that are letters in other alphabets.
5 The universal character name construct provides a way to name other characters.

5.2.1.1 Multibyte characters

1 The source character set may contain multibyte characters, used to represent members of the extended character set. The execution character set may also contain multibyte characters, which need not have the same encoding as for the source character set. For both character sets, the following shall hold:
— The basic character set~~, @ (U+0040 Commercial At), $ (U+0024 Dollar Sign), and ‘ (U+0060 Grave Accent, "Backtick")~~ shall be present and each character shall be encoded as a single byte.
— The presence, meaning, and representation of any additional members is locale-specific.
— A multibyte character set may have a state-dependent encoding, wherein each sequence of multibyte characters begins in an initial shift state and enters other locale-specific shift states when specific multibyte characters are encountered in the sequence. While in the initial shift state, all single-byte characters retain their usual interpretation and do not alter the shift state. The interpretation for subsequent bytes in the sequence is a function of the current shift state.
— A byte with all bits zero shall be interpreted as a null character independent of shift state. Such a byte shall not occur as part of any other multibyte character.

6.4.3 Universal character names

Syntax

1. *universal-character-name*:
    `\u` *hex-quad*
    `\U` *hex-quad hex-quad*

hex-quad:
    hexadecimal-digit hexadecimal-digit hexadecimal-digit hexadecimal-digit

Constraints

2. A universal character name shall not designate a code point where the hexadecimal value is:
~~— less than 00A0 other than 0024 ($), 0040 (@), or 0060 (`);~~
— in the range D800 through DFFF inclusive; or
— greater than 10FFFF

A universal-character-name outside the c-char-sequence of a character constant, or the s-char-sequence of a string-literal shall not designate a control character or a character in the basic character set.

Description

3. Universal character names may be used in identifiers, character constants, and string literals to designate characters that are not in the basic character set.

Semantics

4 The universal character name \Unnnnnnnn designates the character whose eight-digit short identifier (as specified by ISO/IEC 10646) is nnnnnnnn. Similarly, the universal character name \unnnn designates the character whose four-digit short identifier is nnnn (and whose eight-digit short identifier is 0000nnnn).

Acknowledgments

Thanks to Aaron Ballman and Robert C. Seacord for their feedback on this paper!

Reference

Steve Downey - P2558R0 Add @, $, and ` to the basic character set
Philipp Klaus Krause - N2701: @ and $ in source and execution character set