Lifting artificial restrictions on universal character names

Document #:	P2620R1
Date:	2022-08-10
Programming Language C++	
Audience:	EWG, SG-22

Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Abstract

We propose to lift restrictions on *universal-character-names* in identifiers.

Revisions

Revision 1

- Fix typos
- Improve the wording by removing handling of UCNs from phase [lex.charset].

Motivation

There are restrictions on the constitution of *universal-character-names* that seem artificial, and we should lift them[1]!

```
\N{LATIN CAPITAL LETTER I} = 42; // ERROR: I is in the basic character set
\N{LATIN CAPITAL LETTER I WITH DOT ABOVE} = 42 ;// Ok
```

This is by no mean a major issue in C++, as we don't put restrictions on *universal-character-names* in string literals (unlike C), but it is somewhat inconsistent with the lexing model.

Instead of restricting *universal-character-names* values, we can instead mandate that they are part of valid identifiers outside of strings.

Comparison With C

C does not allow *universal-character-names* to designate elements of the basic character set:

> 2 A universal character name shall not designate a codepoint where the hexadecimal value is: - less than 00A0 other than 0024 ($), 0040 (@), or 0060 (‘

[1] This is a small cleanup that isn't worth doing unless we can spend very little time on it, classified as low priority bucket 72.

```
);
```

This has been a pain point for users who would like to consistently use `\u` in string literals as part of code generation processes.

- LLVM issue: Unicode string literals
- Why C99 has such an odd restriction for universal character names?
- Restrictions to Unicode escape sequences in C11

I hope that both languages regain consistency by:

- Not restricting UCNs in string literals
- Not putting restrictions on UCNs in identifiers beyond what naturally falls out of the grammar of identifiers.

Wording

🔲 Separate translation [lex.separate]

4. The source file is decomposed into preprocessing tokens and sequences of whitespace characters (including comments). A source file shall not end in a partial preprocessing token or in a partial comment. Each comment is replaced by one space character. New-line characters are retained. Whether each nonempty sequence of whitespace characters other than new-line is retained or replaced by one space character is unspecified. ~~As characters from the source file are consumed to form the next preprocessing token (i.e., not being consumed as part of a comment or other forms of whitespace), except when matching a *c-char-sequence*, *s-char-sequence*, *r-char-sequence*, *h-char-sequence*, or *q-char-sequence*, *universal-character-name* s are recognized and replaced by the designated element of the translation character set.~~

The process of dividing a source file's characters into preprocessing tokens is context-dependent. [*Example:* See the handling of < within a `#include` preprocessing directive. — *end example*]

🔲 Character sets [lex.charset]

A *universal-character-name* designates the character in the translation character set whose UCS scalar value is the hexadecimal number represented by the sequence of *hexadecimal-digit* s in the *universal-character-name*. The program is ill-formed if that number is not a UCS scalar value. ~~If a *universal-character-name* outside the *c-char-sequence*, *s-char-sequence*, or *r-char-sequence* of a *character-literal* or *string-literal* (in either case, including within a *user-defined-literal*) corresponds to a control character or to a character in the basic character set, the program is ill-formed.~~ [*Note:* A sequence of characters resembling a *universal-character-name* in an *r-char-sequence* does not form a *universal-character-name*. — *end note*]

Identifiers [lex.name]

identifier:
identifier-start
identifier identifier-continue

identifier-start:
nondigit
an element of the translation character set of class XID_Start
universal-character-name
designating an element of the translation character set of class XID_*Start*

identifier-continue:
digit
nondigit
an element of the translation character set of class XID_Continue
universal-character-name
designating an element of the translation character set of class XID_*Continue*

nondigit: one of
`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`
`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 _`

digit: one of
`0 1 2 3 4 5 6 7 8 9`

The character classes XID_Start and XID_Continue are Derived Core Properties as described by UAX #44.

*universal-character-name*s are replaced by the designated element of the translation character set.

The program is ill-formed if an *identifier* does not conform to Normalization Form C as specified in ISO/IEC 10646. [*Note:* Identifiers are case-sensitive. — *end note*] [*Note:* In translation phase 4, *identifier* also includes those *preprocessing-token* s differentiated as keywords in the later translation phase 7. — *end note*]