C++ Identifier Syntax using Unicode Standard Annex 31
The Emoji Problem
Script Issues
Other adopters
That C++ identifiers match the pattern

\[(XID\_Start + _) + XID\_Continue^*\].

That portable source is required to be normalized as NFC.
That using unassigned code points be ill-formed.
Allowed characters include those from U+200b until U+206x; these are zero-width and control characters that lead to impossible to type names, indistinguishable names and unusable code & compile errors (such as those accidentally including RTL modifiers).
STATUS QUO: WE ALLOW OTHER "WEIRD IDENTIFIER CODE POINTS"

- The middle dot · which looks like an operator.
- Many non-combining "modifiers" and accent marks, such as´ and ¨ and .
- "Tone marks" from various languages, including Ꞩ (similar to a box-drawing character ├ which is an operator).
- The "Greek question mark" ; (see below)
- Symbols which are simply not linguistic, such as ☯ and ©.

https://gist.github.com/jtbandes/c0b0c072181dcd22c3147802025d0b59#weird-identifier-code-points
• Follows the same principles as originally used for C++
• Actively maintained
• Stable
• Unicode database defined properties
• Closed under normalization for all four forms
• Once a code point has the property it is never removed
• Roughly:
  ▪ Start == letters
  ▪ Continue == Start + numbers + some punctuation
THE EMOJI PROBLEM

- The emoji-like code points that we knew about were excluded
- We included all unassigned code points
- Status Quo Emoji 'support' is an accident, incomplete, and broken
STATUS QUO IS BROKEN
### SOME STATUS QUO EXAMPLES

<table>
<thead>
<tr>
<th>Not Valid</th>
<th>Valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>int ☢️ = 0;</td>
<td>int ☰ = 0;</td>
</tr>
<tr>
<td>int ☠️ = 0;</td>
<td>int ☠️ = 0;</td>
</tr>
<tr>
<td>int ☟️ = 0;</td>
<td>int ☟️ = 0;</td>
</tr>
<tr>
<td>int ✈️ = 0;</td>
<td>int ✨ = 0;</td>
</tr>
<tr>
<td>int ☹️ = 0;</td>
<td>int 😊 = 0;</td>
</tr>
</tbody>
</table>

When the character was added to Unicode controls validity
Gendered variants of emoji are selected by using a zero width joiner together with the male and female sign.

```cpp
// Valid
bool 👷 = true; // Construction Worker

// Not valid
bool 👷 = false; // Woman Construction Worker (👨‍👩‍👧‍👦)
```
PROBLEMS ADDING EMOJI AS IDENTIFIERS
EMOJI ARE COMPLEX

- Not just code points
- Need grapheme cluster analysis
- May incur costs even for code not using emoji
From the emoji spec

\[
is\text{Emoji}( £ )=false\ for\ Emoji\ Version\ 5.0,\ but\ true\ for\ Version\ 11.0.
\]

It is possible that the emoji property could be removed.
The unicode standard provides a regex that will reject non-emoji, but does not guarantee a valid emoji sequence.

It's not clear how much of the unicode database would be required for complete support.

UNICODE EMOJI
SOME SURPRISING THINGS ARE EMOJI

\( \text{002A} \); Emoji \( \text{# E0.0 [1] (*) asterisk} \)
\( \text{0030..0039} \); Emoji \( \text{# E0.0 [10] (0..9) digit zero..digit nine} \)

\{DIGIT ONE\}{VARIATION SELECTOR-16}{COMBINING ENCLOSING KEYCAP} 1

\{ASTERISK\}{VARIATION SELECTOR-16}{COMBINING ENCLOSING KEYCAP} *

// would this be valid?
int [1] = 1;
Fixing the emoji problem would mean being inventive

Being inventive in an area outside our expertise is HARD

Adopting UAX31 as a base to move forward is conservative

UAX 31 is a known good state
Some scripts require characters to control display or require punctuation that are not in the identifier set.
Apostrophe and dash
- won't
- can't
- mustn't
- mother-in-law

Programmers are used to this and do not notice
Status quo allows these invisible characters

```c
int tmp = 0;
int tmp = 0;
```

- clang 10 warns

```
<source>:2:6: warning: identifier contains Unicode character <U+200D>
that is invisible in some environments [-Wunicode-zero-width]

int t<U+200D><U+200D>mp = 0;
```
However zero width joiner and non joiner are used in some scripts

Farsi word "names"

نامهای

NOON + ALEF + MEEM + HEH + ALEF + Farsi Yeh

نامهای

Farsi word "a letter"

نامهای

NOON + ALEF + MEEM + HEH + ZWNJ + ALEF + Farsi Yeh

نامهای

Anecdotally, these issues are understood and worked around
Identifiers can be checked for what script the code points in the identifier are used, and the rules for allowed characters can be tailored. This requires a Unicode database and would require extensive analysis during lexing. SG 16 does not recommend this.
OTHER ADOPTERS

- Java (https://docs.oracle.com/javase/specs/jls/se15/html/jls-3.html#jls-3.8)
- Python 3 https://www.python.org/dev/peps/pep-3131/
- Erlang https://www.erlang.org/erlang-enhancement-proposals/eep-0040.html
- JS https://tc39.es/ecma262/