Alternative operator representations

C++ (and C) source code may be written in any non-ASCII 7-bit character set that includes the ISO 646:1983 invariant character set. However, several C++ operators and punctuators require characters that are outside of the ISO 646 codeset: `{, }, [, ], #, \, ^, |, ~`. To be able to use character encodings where some or all of these symbols do not exist (such as the German DIN 66003), C++ defines the following alternatives composed of ISO 646 compatible characters.

Alternative tokens

There are alternative spellings for several operators and other tokens that use non-ISO646 characters. In all respects of the language, each alternative token behaves exactly the same as its primary token, except for its spelling (the stringification operator can make the spelling visible). The two-letter alternative tokens are sometimes called "digraphs". Despite being four-letters long, `%:%:` is also considered a digraph.

Primary	Alternative
&&	and
&=	and_eq
&	bitand
|	bitor
~	compl
!	not
!=	not_eq
||	or
|=	or_eq
^	xor
^=	xor_eq
{	<%
}	%>
[	<:
]	:>
#	%:
##	%:%:

Trigraphs (removed in C++17)

The following three-character groups (trigraphs) are parsed before comments and string literals are recognized, and each appearance of a trigraph is replaced by the corresponding primary character:

Primary	Trigraph
{	??<
}	??>
[	??(
]	??)
#	??=
\	??/
^	??'
|	??!
~	??-

Because trigraphs are processed early, a comment such as `// Will the next line be executed?????/` will effectively comment out the following line, and the string literal such as `"Enter date ??/??/??"` is parsed as `"Enter date \\??"`.

Notes

The characters `&` and `!` are invariant under ISO-646, but alternatives are provided for the tokens that use these characters anyway to accommodate even more restrictive historical charsets.

There is no alternative spelling (such as `eq`) for the equality operator `==` because the character `=` was present in all supported charsets.

Compatibility with C

The same words are defined in the C programming language in the include file `<iso646.h>` as macros. Because in C++ these are built into the language, the C++ version of `<iso646.h>`, as well as `<ciso646>`, does not define anything. The non-word digraphs (e.g `<%`), however, are part of the core language and can be used without including any header (otherwise, they would be unusable on any charset that lacks `#`).

Keywords

`and`, `and_eq`, `bitand`, `bitor`, `compl`, `not`, `not_eq`, `or`, `or_eq`, `xor`, `xor_eq`

Example

The following example demonstrates the use of several alternative tokens.

Run this code

```
%:include <iostream>

struct X
<%
    compl X() <%%> // destructor
    X() <%%>
    X(const X bitand) = delete; // copy constructor
    // X(X and) = delete; // move constructor

    bool operator not_eq(const X bitand other)
    <%
       return this not_eq bitand other;
    %>
%>;

int main(int argc, char* argv<::>)
<%
    // lambda with reference-capture:
    auto greet = <:bitand:>(const char* name)
    <%
        std::cout << "Hello " << name
                  << " from " << argv<:0:> << '\n';
    %>;

    if (argc > 1 and argv<:1:> not_eq nullptr)
        greet(argv<:1:>);
    else
        greet("Anon");
%>
```

Possible output:

```
Hello Anon from ./a.out
```

References

- C++23 standard (ISO/IEC 14882:2024):

    - 5.5 Alternative tokens [lex.digraph]
- C++20 standard (ISO/IEC 14882:2020):
    - 5.5 Alternative tokens [lex.digraph]
- C++17 standard (ISO/IEC 14882:2017):
    - 5.5 Alternative tokens [lex.digraph]
- C++14 standard (ISO/IEC 14882:2014):
    - 2.4 Trigraph sequences [lex.trigraph]
    - 2.6 Alternative tokens [lex.digraph]
- C++11 standard (ISO/IEC 14882:2011):
    - 2.4 Trigraph sequences [lex.trigraph]
    - 2.6 Alternative tokens [lex.digraph]
- C++03 standard (ISO/IEC 14882:2003):
    - 2.3 Trigraph sequences [lex.trigraph]
    - 2.5 Alternative tokens [lex.digraph]
- C++98 standard (ISO/IEC 14882:1998):
    - 2.3 Trigraph sequences [lex.trigraph]
    - 2.5 Alternative tokens [lex.digraph]

See also

C documentation for **Alternative operators and tokens**