Ansi Document: **X3J16/93-0052**
ISO Docuemnt: **WG21/N0259**
Date: March 26, 1993
Project: Programming Language C++
Reply to: John Max Skaller
Internet: maxtal@extro.ucc.su.oz.au
Compuserve: 100236,1703
(100236.1703@compuserve.com)

A Proposal to allow Binary Literals, and some other small changes to Chapter 2: Lexical Conventions.

Abstract

(1). Binary integral and character literals of the form

```
0y0100101        0y11011u       0y00100L
'\y01001'        L'\y00101000'
"ab\y10100cd"    L"ab\y11011cd"
```

are allowed to supplement hexadecimal literals for low-level programming. The alternative or supplemental character 'b' is also suggested for the non-character form.

(2) The use of underscores to separate digits in numeric literals

```
4_123_667.33
0y0000_0010
```

is also proposed.

(3) The use of a decimal escape '\d27' for characters is proposed.

(4) A digraph 'addrof' is proposed as an alternative to 'bitand' for clarity when used as the address of operator.

Goal

The lexicology of a language is an important part of its 'look and feel', and strong reactions are evoked on issues of layout and style. Layout in particular has a strong impact on a programs readability, and readability is crucial to the ability to verify program correctness.

In this paper I present some minor enhancements to the C++ lexicology, all of which are devoid of any semantic content, and all of which are trivial to implement. While the benefits will be considered small by some, the cost of implementation is extremely low, and the learning time small enough that there is no real complexity added to the language.

If there are any serious objections to these proposals, I would expect them to be of the same nature of the proposal itself: argument that a particular lexical form might be misleading or error prone should be considered of greatest weight.

C and C++ are often used for machine level programming. It would be a great convenience to developers and maintainers to be able to specify constants in binary.

The specifications of machine level facilities such as the layout of IO ports is often given in reference manuals for the hardware as a binary layout.

Converting this to hexadecimal is error prone, and it is not easy to verify the correctness of hexadecimal constants against binary specifications.

Furthermore, while programming at a low level one often thinks in terms of bits and bit patterns.

In addition, binary constants may be useful when dealing with packed information, and bitmapped graphics such a mouse cursors.

Proposal 1.

The format for a binary integral constant is proposed as

```
0y01000101
0Y01000100
```

and for characters

```
'\y0100101'
```

This proposal automatically allows the binary escape in string literals according to 2.9.4.

```
"ab\y1010010cdef"
```

Alternative 1

The form

```
0b01100001
0B01000010
```

using the letter b (upper or lower case) seems to be more in keeping with existing practice. At least one vendor (Zortech) uses this form. I therefore recommend this form be adopted, despite the fact that 'b' cannot be used for character constants. This form could be adopted in addition to or instead of the form using 'y' for non-character literals.

Alternative 2

The character form can easily be dropped. The form:

```
char(0b0100010)
```

can easily be used where a single character is required. It is not so easy in string literals, but the utility of binary constants in string literals is questionable anyhow. In this case, the use of 'b' for non-character constants seems best, since 'y' was chosen only because 'b' could not be used for character escapes.

Proposal 2

In order to improve the spacing of all non-character numeric literals, binary, decimal, hex or octal, I propose we consider allowing the underscore character between any two digits.

```
4_764_882_376L                  // 4 gigabytes approx
9_123_789.88                    // 9 million approx
3.756_998_11E-9                 // float with 9 digits
0b10111_0000_1111_1111UL        // binary, in nibbles
0b000_1011_0                    // bits 4 to 1 used
0_77567777                      // octal
```

Only a single underscore is allowed between digits, and it is not allowed at the start or end of a string of digits. This form is particularly useful for distinguishing octal notation. The form is not allowed for character constants.

There is some prior art, various lanuages allow optional separators in numeric literals for clarity and alignment purposes.

Optionally, the restrictions on the use of underscores could be eased to allow an underscore between a leading x,X,b or B and the next digit, and after the last digit and before the suffixes l,L,u or U:

```
0x_3f7a
0b_1011_1111
12377_UL
```

which also seems desirable.

Proposal 3

It was pointed out that there is no way to specify a character constant with a decimal form. Either hex or octal must be used. This could easily be fixed by allowing

```
'\d27'          // escape
```

I see no real reason not to do this too.

Proposal 4

The digraph 'bitand' is available and is replaced by & in translation phase 3. However the use of this digraph for the address of operator will be misleading:

```
int *x = bitand y;  // misleading
int *x = addrof y;  // better
```

so I propose 'addrof' as an additional digraph for &.

Compatibility

No conforming C code will be broken by these proposals, except that the digraph 'addrof' will necessarily break programs using 'addrof' as an identifier. Another digraph might be chosen that is considered more suitable.
No existing C++ will be broken by these proposals (except for the digraph).

Explanation of choice of lexicology

Note that the use of '\z' where 'z' is not one of the defined characters (ntvbrfa\?'"x0123456789) is currently undefined.

The choice of y (as in binar*y*) was made because:

- y is close to x(hex) in the alphabet
- b,n,r are reserved for character constants for backspace, newline and return
- a and b are also a hex digits.

That left 'i' and 'y', of which I think 'y' is better. For the non-character constants, b could replace or supplement 'y'. If binary character literals are not accepted, then b is the prefered letter for binary integral literals.

Explanation of no binary IO

There is no proposal to provide a binary facility for the printf or related functions. Hex input and output is available, and it is easy to code a function that does binary I/O. Such a facility would break backwards compatibility with the standard C library functions. The proposed bitset classes might also be used for binary IO.

Relationship to Bitset classes.

The proposed bitset class cannot replace proper binary literals. Although the forms

```
bits("00010010").to_uint()
bits(0b00010010).to_uint()
0b00010010u
```

are equivalent semantically, the first is sure to be slower and is checkable only at run-time. Even with inline functions, the third form is shorter and more readable than the second form. However, the bitset classes no doubt have advantages, and I can only see that allowing binary literals can do nothing but enhance the utility of the bitset classes.

Editorial Changes required - Binary literals

The following editorial changes are required to accommodate this proposal:

2.9.1/1: Append:

"A sequence of digits preceded by 0b or 0B is taken to be a binary integer (base 2). The binary digits are 0 and 1."

2.9.1/2: replace

"If it is octal or hexadecimal"

with

"If it is binary, octal or hexadecimal"

2.9.2/2: Add to the table

"binary number bbbbbbbb \ybbbbbbbb"

2.9.2/3: Add in the approriate places:

"The escape \ybbbbbbbb consists of a backslash followed by y followd by a sequence of binary digits that are taken to specify the value of the desired character."

and

"There is no limit to the number of binary digits in the sequence"

Change:

"A sequence of octal or hexadecimal digits is terminated ..."

to read

"A sequence of binary, octal or hexadecimal digits is terminated ..."

Opinions of programmers

Below are a collection of opinions collected in a few days from comp.lang.c++ in response to a request to speak up in support of, or against, binary literals.

I said:

>If you WANT binary literals, and can think of a good use for them,
>now is the time to speak up.

>The proposal will almost certainly be rejected as too unimportant
>to bother with otherwise. (IMHO).

>(If you object, also speak up)

and posted the original proposal without the section on editorial changes, along with the comment:

> This proposal has been changed to use
>
> 0b01100101
>
> instead of
>
> 0y00110101

Here are some of the responses.

: This proposal has been changed to use
:
: 0b01100101
:
: instead of
:
: 0y00110101

Much more sensible!

: If you WANT binary literals, and can think of a good use for them,
: now is the time to speak up.

They're more use than octal, and it seems odd to have decimal/octal/hex
and no binary. Can make tings much clearer if you're actually dealing
with bit patterns.

>If you WANT binary literals, and can think of a good use for them,
>now is the time to speak up.
>
Yep ! - why not !? It must be very simple to implement...

Main reason to use binary litterals should be for clarity
- which is not unimportant !???

I've found myself wanting binary constants very often,
and I hate figuring out hexadecimal equivalents, when it's just a small
extension to the language which is needed.

Sounds like a great idea to me. It's frustrating to have to convert
to hex or octal, especially in situations where the bit positions
have meaning. I have a related question, too. Why isn't there
a way to convert a bit field specification of, say, an unsigned
int into an actual unsigned int that you can use real bit operations
with, such as ^, |, <<, ...?

I would *love* to have binary literals. With all the bit-twiddling
that goes on, even in C, and the claim that this low-level bit
twiddling is a bonus for the language, I can't understand why its not
there already! :-)

I like your proposal for the "binary" constant type. I think, however,
that you may find it necessary to provide printf() functionality;
I would doubt that the committee will look favorably on any
proposal that adds yet another "special case" to the language.

Thought I'd take this off line to avoid the hornets on the reflector. I
like your binary constants proposal, and would vote for it in full
commitee, but am not usually too active in the extensions group.

I'm not sure if we met in Portland or before, or how long you've been
on this commitee, but it sure seems you stirred up a hornets nest. Of
course your critics are right that voting NO is easy, and the burden of
proof is always on the proposer. Perhaps you should approach Zortech to
see what their experience is, and whether they might submit the
proposal themselves. Another piece of existing practice is the
Macintosh resource compiler, which uses a C syntax extended with binary
constants (as I recall 0bxxxx). Blessing an existing practice at the
request of the vendor is bound to be easier than proposing a new
practice.

I like the idea, but I doubt it'll fly since it's clearly small, totally
backward-compatible, and may actually be useful. Unfortunately, a macro
defined to a hex constant is a usable workaround for binary constants:

```
#define B01101110 0x6E
```

so you may have a tough time getting it through the committee.

On the whole, I'd prefer 0y101011 for constant expressions to be consistent with the \y10110 notation, but I can understand going with 0b10110 since it would be much more commonly used than the \y character constants.

You ask:

> What is it I have to prove that is provable?

Proof is the wrong word. But basically noone needs to be convinced to vote NO: they need reasons to vote YES. And they might vote NO anyway.

> Yes. I expect other vendors to provide binary constants
> *anyhow*. Better to standardise the way its done now, before they
> all provide a different syntax.

For the commitee this would be the best sort of reason, if other vendors plan to follow Zortech's lead. For me the best reason is that I hate doing binary to hex conversions, and can't do it at all without a calculator. But I still need to twiddle bits.

I wouldn't bother with it. At the point I'm programming registers, I'm usually debugging in hex anyway, and would prefer the source to be in hex.

However, it looks less harmful than trigraphs ...

It is easy to allow literals in arbitrary bases from 2 to 36.

```
2x01100101 /* binary */
8x145      /* octal, same as 0145 */
10x101     /* decimal, same as 101 */
16x65      /* hexadecimal, same as 0x65 */
```

From: c++std=ext@research.att.com
Original-From: "Larry Thiel" <lthiel@denitqm.mnet.uswest.com>
Subject: Re: Binary Constants

I have to agree with Martin O'Riordan (and others) that there is minimum benefit and minimum cost to the proposal. Martin mentioned that previous discussion had not mentioned glyphs or other graphic uses and I will add it had also not explicitly pointed out the value of graphic representation of hardware switches, port contents, bitmapped flags, etc.

Martin recognizes that using a comment to point out the intent cannot be checked by the compiler, but perhaps does not recognize just how important that is to some people.

The bitset proposal addresses those needs but with a complicating factor those people don't need and may not appreciate. One also wonders if those people dealing in realtime hardware interfaces, transport protocols,

etc. might be impacted by conversion overheads from new types to
built-in integral forms (especially when re-using pre-existing code).

I am not a strong supporter of the proposal but neither am I an opponent.
If the cost is truly as minimal and the work is as minimal as people
seem to think then I would support it.

BTW, given the expectation that mainstream users would (hopefully)
use bitsets most of the time, I have no particular preference
about the syntax (i.e 0y0101 vs 0b0101 or both).
In fact, I might wonder whether the character form needs
support for the set of users I am referring to.
Perhaps someone else could comment on that.

>>If you WANT binary literals, and can think of a good use for them,
>>now is the time to speak up.
>>
>Yep ! - why not !? It must be very simple to implement...

It sure is easy to implement. And I had numerous cases where a bit constant
would have been much clearer than a hex constant. Especially '1001', etc.,
which looks very strange as 011, 0x9, or 9U.

fixed bit values are common in compression and communication. I want them!

However, I don't see a strong need for character bit constants.
If anything, I would like to see DECIMAL CHARCTER CONSTANTS. Why do
I have to say '\x1b' when everyone knows that 27 is ESC ?!
I can easily type ALT+139 to see decimal char 139 on my PC. I have to go
to a table to find what '\x8b' is.

In any case, I'm for bit constants!