Comment on Proposed Trigraph Deprecation

Michael Wong, Hubert Tong, Robert Klarer, Ian McIntosh, Raymond Mak, Christopher Cambly, Alain LaBonté
michaelw,hstong,klarer,ianm,rmak,ccambly@ca.ibm.com,
alabon@gmail.com

Document number: N2910=09-0100
Date: 2009-06-19
Project: Programming Language C++, Core Working Group
Reply-to: Michael Wong (michaelw@ca.ibm.com)
Revision: 1

Comment on Proposed Trigraph Deprecation

1 Abstract

In the Summit C++ meeting, we promised to look into whether trigraphs are commonly used on software that is normally not found through Google Code Search based on UK 11, before we formulate an opinion on trigraph deprecation. We have conducted such research and concluded that there is existing code from world-wide companies (other then IBM) that use trigraphs. Some of these users have commented that they will be intensely unhappy with the deprecation of trigraphs, as they may have no alternative.

As such, Canada intends to vote NO to the deprecation of trigraphs, because we have learned since the Summit meeting that trigraphs are commonly used by a large and important segment of the C++ community. We do not consider the deprecation of such a widely used language feature that is crucial to so many C++ users to be appropriate. We urge others to carefully consider their position on Trigraph Deprecation.

2 Overview

Our intention is to oppose trigraph deprecation as proposed by UK11 and CWG 789. This paper will outline some of the technical issues. This deprecation as described by UK 11 is basically a desire to increase support of novice users by reducing a source of potential confusion, and that there seems to be no use of trigraphs through Google Code Search. We have verified that there is a large usage base for trigraphs.

We are convinced that as a Standard body, the single most powerful argument against trigraph deprecation is not any technical matter (although there are many); but that it is

we need to continue to support existing users, over the apparent syntactic inelegance to novice users.

In the words of existing trigraph users from a world-wide company:

“In that capacity we have widely distributed deployments, around the world, across Windows, HP-UX, Linux, z/OS and iSeries systems. That's why we need the trigraphs, it doesn't seem you can count on the [ ] characters working everywhere. ... "

And from a consulting firm:

“…In all our header files. That includes libraries like boost. The bottom line is we choose IBM-1047 as the base codepage because it's the best match for ISO-8859-1 latin-1. I have opened problems with the likes of doxygen to fix missing trigraph support and had no problems. Same with Enterprise Architect etc. Pre-processing is easy, why drop something which will break something”

3 History

In the Summit C++ Standard meeting, CWG was processing National Body Comments and encountered UK 11:

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2837.pdf

One of the comments from UK 11 was to deprecate trigraphs.

“Trigraphs are a complicated solution to an old problem, that cause more problems than they solve in the modern environment. Unexpected trigraphs in string literals and occasionally in comments can be very confusing for the non-expert.”

Because trigraph deprecation has the effect of a fix to the C++0x CD1 (Committee Draft 1), a Core language issue was opened and now drafting is being prepared, which will likely be voted in Frankfurt.

http://www.open-std.org/jtc1/sc22/wg21/docs/cwg_active.html#789

4 The real issue

While there are significant technical issues with trigraph deprecation, we have relegated them to Appendix A for those who wish to study them in detail. We have chosen to focus on the matter of supporting existing users who have massive investment into trigraphs for one reason or another.

The biggest argument against deprecating trigraph is the fact that we know there are source code from world-wide companies, based on an internal survey of our customer

problem reports that make heavy use of trigraphs for one reason or another, and customer testimonials. Google code search doesn't find them because these are proprietary code. Further discussions with some of these customers have shown serious dissatisfaction with WG21's intention to deprecate trigraphs, because they do not wish to make any changes to their source code. It doesn't seem fair to force this group of existing users to change their source code for no apparent benefit, other then reducing an annoyance to another group.

Up to now, there has been enthusiastic support for trigraph deprecation. There may be diverse reasons. The reason as stated by UK11 is because non-expert C++ users can easily confuse trigraphs (which start with ??) with a question mark in strings. There may also be a wish to make text easier to automatically be searched, and indexed without such confusion (as with a search engine). Others felt that they are simply not used or that they damage the language in some way. There may be additional reason that we are not aware of.

But the fact that there are real current users who use trigraph, and the unlikelihood that C will ever remove trigraph, are in our opinion, powerful arguments against trigraph deprecation over the inconvenience of getting some confusion for non-expert users (paraphrased from UK 11) because the Standard is supposed to serve the complete existing user base. Depending on the size, complexity of the code, current users of trigraph will resent the need to change their code which may be error prone, especially if they want the same file to work in both EBCDIC and non-EBCDIC world, or in the C world, or in strict C++0x Standard compliant mode. Digraph, as shown in the Appendix is not a complete trigraph replacement, especially inside another token such as a quoted string. For the EBCDIC world, there is additional powerful argument where it may even be necessary to use trigraph to prevent code-invariant confusion.

We argue that existing users must continue to be supported under these conditions There are a number of reasons novice users may be avoiding C++ or C++0x. We would argue that trigraph is not a significant one, if it is even one of them.

If the argument is to make text pages easier to search and index by search engines without the confusion, then this becomes an implementer versus user issue. In such case, we believe the user should always win, unless the implementation is very hard. We do not view this as such an exception.

We are choosing to support existing trigraph users, and respectfully urge that a new alternative be considered.

5 The effect of deprecation

At this point, many will point out that deprecation may not mean anything, that compilers will probably continue to support trigraphs anyway, or that this is really just an EBCDIC/IBM problem, and IBM will likely continue to support trigraph as an extension. Why should we burden the rest of the world with this feature?

This current state of CWG 789 is drafting. If completed, it will be voted in as a language fix during the next meeting's Core sub-committee deliberations when the proposed draft text is reviewed, along with many other defect fixes to the language that also has proposed text review. The proposed text change is very simple in this case. It will likely move all of Clause 2.4 to Appendix D. All these fixes will be packaged and voted in full plenary at the end of the week (for which issue 789 will just be one of many) by all national body representatives.

Appendix D lists all deprecated features. It defines deprecated as:
"Normative for the current edition of the standard, but not guaranteed to be part of the Standard in future revisions."

One of the clause (in Appendix D) that has been deprecated are C-headers (like stdio.h), but many compilers (as our compiler) continue to silently support it in some extended mode. But in strict Standard conforming mode, compilers may start to issue some diagnostics, or continue to ignore it. It is hard to say what each compiler's action will be.

Very little is actually ever removed from the Standard. So this, along with many other items may continue to stay in Appendix D for a long time.

So why should it matter that we deprecate trigraphs if there is likely no change to the status quo. If such is the case, why do we bother to deprecate anything?

There is one important consideration and it is also contrary to existing users. Once deprecated, the Standard may be free to replace trigraph with another meaning, and this is definitely not an outcome existing users would want if trigraph deprecation becomes an intermediate step towards replacement of the semantics.

6 Summary

In summary, we highlight the major problems when trigraphs are deprecated:

1. Many real users (and not just IBM) do use trigraphs and resent changing their code for the convenience of another group, or implementers, unless there are really good reasons.
2. There is no replacement for trigraphs in quotes
3. Compiling the same code between ASCII and EBCDIC without #ifdefs
4. Interoperate headers between C with trigraphs and C++ without trigraphs

There are additional specific technical problems for EBCDIC which is one of the main constituent, but we will not list them here as we believe the above report serve as powerful reasons to not deprecate trigraphs.

Alain LaBonté, Convenor of ISO/IEC JTC1/SC35/WG1, ISO/IEC JTC1/SC2/OWG-SORT, Project Editor of several International Standards in JTC1/SC35 and JTC1/SC2, in private communication in response to problems of trigraph deprecation:

“That is good argument that is sufficient to say that deprecation should not be an option.”

Appendix A

Trigraph deprecation is vexing for existing world-wide companies (of which IBM is one such user) because of their use of EBCDIC variant code pages. It makes it difficult to start in a code page neutral manner, establish what the active code page is, then start using invariant characters. We use a pragma filetag("IBM-1047") to toggle the code page, then we can use #. Prior to that we are in a code page neutral environment and must use the trigraph ??=

```
??=ifndef _DEQUE_T
??=if defined(__MVS__)
??=if defined(__COMPILER_VER__)
??=pragma filetag("IBM-1047")
#endif /* defined(__COMPILER_VER__) */
```

Trigraph '??=' is used at the beginning of the source because the code point for '#' differs across various EBCDIC code pages. And without the filetag pragma, the compiler assumes the codepage is IBM-1047 (the default EBCDIC codepage). '?' share the same code point across all EBCDIC code pages, and so does '='. Therefore the trigraph is used to represent '#'. By the same token, the digraph %: also has the same property, so it may be reasonable to replace the existing trigraph with digraphs. I will show later that this too is not enough in the context inside other tokens.

Another example, the following code is portable across all EBCDIC codepages:

```
int main ()
??<
int a = ??/
3;
return a;
??>
```

This seems to be a sound argument for NOT deprecating trigraphs, because characters like '\' is only representable using trigraphs.

First, digraphs are not a complete replacement for trigraphs.
The digraph characters are:

%:	#	number sign
<:	[	left bracket
:>	]	right bracket
<%	{	left brace
%>	}	right brace
%:%:	##	

The trigraph sequences are:

??=	#	pound sign
??(	[	left bracket
??)	]	right bracket
??<	{	left brace
??>	}	right brace
??/	\	backslash
??’	^	caret
??!	|	vertical bar
??-	~	tilde

Digraphs do not represent the following:

??!	|
??'	^
??/	\
??-	~

The last 4 trigraphs are not represented by any digraph. This is a problem as a replacement.

Even if digraphs were a complete replacement, they do have differences. Unlike trigraphs, digraphs are handled during tokenization, and must always represent a full token by itself. If a digraph sequence occurs inside another token, such as a quoted string, or a character constant, it will not be replaced. Trigraphs are handled during phase 1 of translation and do get replaced even inside another token. In fact, this is one of the big reason why trigraphs are disliked because inside strings, they get confused with simple '?' characters before tokenization.

Except for backslash, we can also use the alternative spellings of operators as replacements:

bitor
or
or_eq
xor
xor_eq
compl

The backslash truly has no alternative for use in universal character names, strings and character literals except for the trigraph.

Acknowledgement

This paper was produced with the help of internal research and some external discussions with Stehen Michell, Uma Umamaheswaran, Tetsuji Orita, Seiji Hayashida, Catherine Morton, Sasha Kasapinovic, Wang Chen, Kendrick Wong, Sean Perry, Tim Burrell, Jichao Zhang, Zach Zylawy, Tom Schmidt, Rene Matteau, Basil Kanneth, Pete Balm, David Crayford, Tim Hahn, Leon Kiriliuk.