Title: IBM comment on preparing for a Trigraph-adverse future in C++17

Number: N4210
Date: 2014-10-10
Authors: Michael Wong, michaelw@ca.ibm.com
Hubert Tong, hstong@ca.ibm.com
Rajan Bhakta, rbhakta@ca.ibm.com
Derek Inglis, inglis@ca.ibm.com
Project: Programming Language C++
Reply to: Michael Wong michaelw@ca.ibm.com (Chair of SG5)
For: EWG
Revision: 1

Introduction

In Rapperswil, EWG had a vote to remove Trigraph based on N3981. The vote was SF 9, WF 6, N 6, WA 3, SA 2. A revote with no neutral option was SF 10, WF 9, WA 3, SA 3. A separate vote on Deprecate/Remove/Status Quo in a three way vote was 5/16/6. The Strongly Against vote was IBM, Canada, and a few others. IBM and any users with EBCDIC code are the most affected by this removal in terms of losing portability. That is still the main reason for the existence of trigraphs in C++. In an ASCII world, they are an annoyance because they get quietly replaced in quoted strings, such that strange combinations of leading ?? with any of =,(,),<,>,/,’,!,and – can become surprisingly replaced with some single character.

IBM argued against this removal on two points. One was that this was a rehash of previous attempt to deprecate trigraphs in 2009 which failed and with no new information presented, this was wasting committee time. Two was that that this makes C++ less portable in an ASCII-centric world where EBCDIC is still an important, although a smaller player using C++. While a few members heard these arguments loud and clear, there was clear consensus that C++ is becoming an ASCII-dominated language where the annoyances of the many should outweigh the needs of the few. In some way, there is language design reality in that too as we need to ensure C++ remain friendly to the majority. The only question is how large is that majority and how small is that minority.

This feature was approved in EWG, and CWG and while it was not voted in plenary (due to C++14 being in flight), it is effectively approved modulo a small lifeline offered by EWG.

IBM was asked by Bjarne to look at how many real users of EBCDIC that use trigraphs there remain and to return to EWG with a position. This paper summarizes that IBM position, give guidance for trigraph users moving forward, and outline IBM future voting plan on this issue. As such, we will tell the story of the Bad, the Good, and the Ugly.

The Bad

All the technical arguments of our 2009 paper N2910 still remain valid.
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2009/n2910.pdf

In that paper, we analyzed the problem, looked in-depth at various solutions and explain why alternatives like digraphs and alternative spelling operators are no replacement for trigraphs.

There are real customers who use EBCDIC. We cannot reveal their names due to confidentiality agreements. One key example is some of the major banks in North America who continue to use IBM machines to perform check clearing operations. These high reliability software systems are written on IBM mainframes clearing your checks and because they have been debugged over so many years and are highly critical to daily integrity of the financial industry, they are highly reliable and will never be moved to any other platform. In a way, this makes their code unaffected by this removal of trigraphs.

Other EBCDIC users with trigraphs from a world-wide company that can't get square brackets (i.e. [ ]) everywhere so must use trigraphs to get at them quotes:
"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. ... "

A major motivation at the time trigraphs were added was to support people with keyboards that lack these characters and we believe this continues today.

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"

The real point of this issue is the underlying assumption that everyone uses ASCII or non-EBCDIC Unicode. If C++ wishes to continue to serve both communities, then you need trigraphs to allow portable programs in at least the cases we know about (EBCDIC). The whole point of a standard is portability, and removing trigraphs removes portability. However, if the premise is not to support EBCDIC, then this changes everything.

To pile on the portability argument even more: even if we were to ignore the EBCDIC world and stay in the ASCII world, mixed compilation of trigraph-dependent (from C or pre-trigraph removal C++ code) and trigraph-adverse source bases (from Open Source codebase after this removal), will become difficult in future without code change by one of the other program (most likely the trigraph-dependent one). This is what this change means. So the obvious rebuttal will be how much is that code, and how many users does that affect? The removal side's opinion is that it is "vanishingly small". I believe neither side will know that number exactly, nor will each side change its mind.

To summarize the bad which is the same as it ever was in N2910, IBM still maintains that removal of trigraphs is bad for the C++ language if we continue to want C++ to work in the EBCDIC world.

The Good

Trigraph removal based on N3981 is relatively benign in that it still allows a valid implementation to retain trigraph support even in the Standard mode, or as an extension in C++17 and after. We believe and strongly urge all implementations to follow this guidance and retain support for trigraphs at least as an option. IBM compilers will likely keep it in the Standard mode. We urge users to support their compiler vendors in retaining this option.

In reality, we realize the world is moving mostly to non-EBCDIC and indeed even the GNU compiler, at least in version 4.7 does not have trigraph support in its default mode (which is an extended mode), for some time now. Trigraphs have to be enabled specifically, or in pedantic mode. We did not check, but believe Microsoft compilers behave the same. Clang 3.5 was released with trigraph removed from Standard mode, but retained as an option after this feature was approved through EWG.

We recognize the removal of trigraph will make C++ less surprising, easier to teach, and possibly improve its adaption for non-experts for a majority of users because the majority are in the ASCII or non-EBCDIC Uncode world, and do not know about the needs of the EBCDIC world. We also agree that raw string, while designed to help fix the surprise replacement annoyance in quoted string is really a pill that is taken the day after it happens and thus not entirely satisfactory.

The Ugly

After significant consultations within IBM, it is IBM's position that for the harmony of the greater C++ community, we will not oppose C++17 because of the removal of trigraphs. We recognize that C++ is mostly an ASCII-centric language now. We will continue to oppose trigraph removal, because we feel someone must speak for the minority of users who cannot speak for themselves. This is not just taking a moral high ground, but being practical. We realize the tide is against the EBCDIC world and as such, whether trigraph is removed or not, IBM compiler, EBCDIC, and non-ASCII users must plan to operate in such a world and it is best to start now.

We like to outline that plan for users. While we feel that the removal of this feature does not specifically harm IBM, we caution that it likely will harm some user somewhere. Whether trigraph is removed or not, IBM and users will need to adapt various interoperability features for its compilers and customers, such as for example to enable fine-grain trigraph control, or to ease mixed inclusion of trigraph-dependent and trigraph-adverse code (such as C++17 when trigraph is removed).

After some consideration, we also do not recommend adaption of any half-way measure such as the following:
1. How about trigraphs only in the first Macro (until the closing macro) token groups?
2. How about disable trigraph replacement in strings?

We feel such half-way measures actually complicate the trigraph situation and complicates C++. It makes it more problematic because users have to know not only whether this feature is on or off, but now have to know where. We feel it is simpler to just have it either be on or off, unless some fine-grain ugly pragma is used to clearly indicate the status.

Conclusion

This is a friendly comment to the whole trigraph removal issue with what we hope is a measure of realism and useful advice to users to adapt to in future.

Acknowledgement

We are grateful for the consultation on this paper with Paul Mckenney, Ian McIntosh, Milos Lalovic, Thomas Petrolino, Raymond Mak, Sarah McNamara, Zbigniew Sarbinowski, Chris Bowler, Youssef Himo, Stephen Michel and many others, whom I may have unintentionally forgotten.