

Proposal for C2y  
WG14 3192

Title: Sequential Hexdigits  
Author, affiliation: Alex Celeste, Perforce  
Date: 2023-11-30  
Proposal category: Clarification/enhancement  
Target audience: Users

## **Abstract**

We propose to enhance the guarantees made by the Standard about the values of character constants so that the promises allowing arithmetic for decimal digits are consistently made available for hexadecimal digits as well.

The proposed change does not appear to have any impact on real implementations and simply makes a guarantee to users about something which was already true.

# Sequential Hexdigits

Reply-to: Alex Celeste ([aceleste@perforce.com](mailto:aceleste@perforce.com))  
Document No: N3192  
Revises: (n/a)  
Date: 2023-11-30

## Summary of Changes

### N3192

- original proposal

## Introduction

In section 5.2.1 “Character sets”, the Standard says that:

In both the source and execution basic character sets, the value of each character after 0 in the above list of decimal digits shall be one greater than the value of the previous.

No other prescription is made for the specific values of any characters.

This is handy, because it very conveniently allows conversion from text to integer value by doing arithmetic against a read character value and '0' or '9', such as the check `(x >= '0' && x <= '9')` to serve as an “is decimal digit” test, and subsequently `val = x - '0'`; to get the value of the character as a decimal digit.

However, because no guarantee is made for any other characters, the equivalent code for the hexadecimal digits A through F is technically non-portable (`((x >= 'a' && x <= 'f') || (x >= 'A' && x <= 'F')), x - 'a', x - 'A'`). This makes extending decimal-handling code (which implicitly covers base-2 and base-8 because the digit ranges are included) to also cover hexadecimal, onerous to do in a strictly conforming way.

## Proposal

We propose that two new clauses are added to the end of the existing sentence in 5.2.1 paragraph 3:

...; the value of each character after a, up to and including f, in the above list of lowercase letters shall be one greater than the value of the previous; and the value of each character after A, up to and including F, in the above list of uppercase letters, shall be one greater than the value of the previous.

A paragraph break for readability may be appropriate. Less repetition to unambiguously say the same thing may be appropriate.

## Impact

As far as we can tell, all character sets in common use already support this guarantee.

The vast majority of character sets in use anywhere are derived by varying degrees from ASCII, which places all 26 Roman lowercase and uppercase characters in two contiguous blocks anyway, so not only does the A-to-F assumption hold, but on the majority of systems A-to-Z is valid as well.

The remaining minority of character sets in use all appear to be variations of EBCDIC, which has gaps in the alphabetic blocks and therefore does not support the A-to-Z assumption, *but* does guarantee contiguity of eight-character blocks, so the EBCDIC invariant subset [1] still guarantees that the Roman characters A-to-H are contiguous and therefore that the hexadecimal assumption holds.

Therefore, we do not expect any implementation should actually need to change, and this is a backwards compatible enhancement to the text that retroactively makes existing user code more portable.

Library code is unlikely to be affected as the Standard library `ctype.h` is more likely to be implemented using other techniques (i.e. lookup) [2]. However, simpler libraries demonstrate the assumption case [3], and even complex implementations treat the assumption as valid without necessarily relying on it [4].

This proposal should therefore also go on the “Papers of Interest” list for implementers wishing to support previous editions of the C Standard, as the assumption is likely to be back-portable.

This proposal does **not** affect the definition of a “hexadecimal digit”, which may include other characters depending on the locale (the C++ `std::isxdigit` is capable of distinguishing these, while C’s `isxdigit` is not). This proposal only touches characters that the Standard already requires to be available and to have some value.

## References

[C23 public draft n3096](#)

[EBCDIC Wikipedia](#)

[glibc ctype.h](#)

[Barracude72 isxdigit](#)

[Clang test for A-to-F](#)