Bits & Bytes

Arkansas' Premier Computer Club

July 2021

Bella Vista Computer Club - John Ruehle Center

Highlands Crossing Center, 1801 Forest Hills Blvd Suite 208 (lower level), Bella Vista, AR 72715

Website: http://BVComputerClub.org **Email: editor@bvcomputerclub.org**

For July we will resume in-person Board and General meetings at the Highlands Crossing Center, 1801 Forest Hills Blvd. Because of upward trends of COVID-19 among the unvaccinated, if you fall into that category or have other family members that do, we still highly recommend observing masking and social-distancing guidelines.

MEETINGS

Board Meeting: July 12, 6pm, in John Ruehle Training Center, Highlands Crossing Center.

General Meeting: July 12, 7pm, "Internet Searching", presented by Joel Ewing. This will cover the basics of web search engines, the most popular search engines, how to tell your browser to default to a different search engine, and ways to refine searches. We will meet in Room 1001 on the lower level of The Highlands Crossing Center, 1801 Forest Hills Blvd, Bella Vista. Visitors or Guests are welcome.

We may experiment with a parallel Zoom broadcast of the meeting, but how well that will work is uncertain. If so, Zoom access information will be published on our website.

Genealogy SIG: **No meeting** (3rd Saturday).

HELP CLINICS

July 21, 9am - noon at John Ruehle center
(none on July 3 holiday weekend)
Members may request Remote Help on our website at https://bvcomputerclub.org at menu path Member Benefits ►Remote Help .

MEMBERSHIP

Single membership is $25; $10 for each additional family member in the same household.
Join on our website at https://bvcomputercub.org at menu path Get Involved ►Join/Renew, by mailing an application (from the web site) with check, or complete an application and pay in person at any meeting.

CLASSES

Thursday, July 22. Computer Security for Regular People, Part 1, 6:30pm – 8:30pm, with Justin Sell. In-person class at the BVCC Training Center. Maximum attendance 8.

Advance sign up required for each listed class: Contact Grace: email to edu@bvcomputerclub.org, text 469-733-8395, call 479-270-1643, or sign up at the General Meeting. Classes are **free to Computer Club members.** Class access information will be emailed to those signed up for the class the day before class.

Check the monthly calendar and announcements for any last minute schedule changes at http://bvcomputerclub.org .

NEW OR RETURNING BVCC MEMBERS

We are pleased to welcome the following new members or members returning to BVCC after an absence since last month's newsletter:

Michelle D'Almeida	Virginia D'Almeida	Mary Doyle
Peter Doyle	Jeanette Faber	Steve Guter
Kathleen Morgan		

STORING NUMBERS IN NUMBER BASES OTHER THAN BASE-10

By Joel Ewing, President Bella Vista Computer Club
president (at) bvcomputerclub.org
Bits & Bytes, July 2021

Why This is Relevant

The number representation everyone is taught in grade school is the decimal notation, using the ten digits 0 through 9, where the value represented by the digit is multiplied by a power of 10 based on its position relative to the decimal point. Progressing to the left the powers of 10 increase ($10^0 = 1$, $10^1=10$, $10^2=100$, etc.); following the same pattern progressing to the right of the decimal point, the powers of 10 decrease ($10^{-1}= 1/10$, $10^{-2} = 1/100$, $10^{-3} = 1/1000$, etc.)

On most computers today, the most commonly used internal representation for numbers that are used in calculations is based on binary or base-2. This internal format is chosen because it is the format that is most efficient for storage space and most directly supported by the hardware, meaning that it also takes the least amount of clock time and CPU resource to perform calculations with numbers in this format.

Computer application users, on the other hand, enter values in decimal or base-10, and expect results to be displayed in decimal. That base-2 is used under-the-covers is for the most part invisible to the user, but there are some cases – specifically when decimal places are involved – where unexpected results can become visible to the user: values that the user expects to be identical and which may even be displayed as identical are not; values displayed to enough decimal places appear to have "random" garbage digits or unexpected "rounding" errors. Occasionally users experiencing these artifacts for the first time will report them as software "errors". They are not errors, but the unavoidable consequence of the use of binary representations internally to improve the performance of the application.

Spreadsheet applications like Excel and Calc store numeric values in base-2, but use enough significant digits so that with rounding of displayed decimal equivalent values the counter-intuitive effects with fractional values are

rarely seen by users[1]. You should, however, be aware that if you started to work with very large currency values accurate to one cent (on the order of trillions of dollars), that you might start seeing discrepancies at the penny level.

Integer Values

Any integer value may be exactly represented using any base ≥ 2, provided storage is available for enough digits. The higher the base, the fewer the number of digit positions generally required to represent larger values.

To represent a value using base n requires n unique digit symbols corresponding to numeric values of 0, 1, …,up to n-1. For decimal base-10 we use the 10 symbols "0", "1", "2", "3", "4", "5", "6", "7", "8", and "9". For base-n where "n" is less than 10, we use the first n digit symbols used for base 10. For hexadecimal base-16, we need 16 symbols, now customarily using "0" through "9" plus "A", "B", "C", "D", "E", and "F" to represent the six values corresponding to the decimal values 10, 11, 12, 13, 14, and 15.

So, for example, the value 510 in base-10 (written as 510_{10} to explicitly show the base used) is interpreted to mean (taking digits from right to left) as $0 + 1 \times 10 + 5 \times 10^2$

In base 2 that value would be $111,111,110_2 = 0 + 1\times2 + 1\times2^2 + 1\times2^3 + 1\times2^4 + 1\times2^5 + 1\times2^6 + 1\times2^7 + 1\times2^8$
= (in base 10) $2 +4+8+16+32+64+128+256 = 510_{10}$

In base 3 it would be $200220_3 = 0 +2\times3 + 2\times3^2 + 0\times3^3 + 0\times3^4 + 2\times3^5$
$= 6 + 18 + 486 = 510_{10}$

In base 8 it would be $776_8 = 6 + 7\times8 + 7\times8^2$
$= 6 + 56 + 448 = 510_{10}$

In base 16 it would be $1FE_{16} = 14 + 15\times16 + 1\times16^2$

$= 14 + 240 + 256 = 510_{10}$

To represent the same numeric value takes 9 digits in base-2, 6 digits in base-3, 3 digits in bases 8, 10, and 16.

The bases most commonly used to represent values stored in a computer are base 2, base 8, base 10, and base 16; although in the past some other unusual bases like base-3 have even been used.

Bases corresponding to a power of 2 have a special relationship to base-2 values in that they may used as a kind of "shorthand" for representing a binary value but using either one third or one fourth the number of digits.

To write a binary value like 11111110_2 as an octal (base-8) value, no elaborate conversion process is required. Simply group the binary digits into groups of three digits (adding extra high-order zeros if not already a multiple of 3 digits) as in (111) (111)(110), treat each group as a separate value and calculate the base 10 value as in (7)(7)(6), and notice that 776_8 is indeed the representation of the same value in base 8. To reverse the process simply

---

1 With some deviations, Excel uses the IEEE 754 standard double-precision binary floating-point numeric format for storing numbers using 64 bits to store values with 53 binary bits of precision and a multiplier of 2^{-1022} to 2^{+1023}. This roughly corresponds to 15 decimal digits of precision and a magnitude-of-value range in decimal from $1.79769313486232x10^{308}$ to $2.2250738585072x10^{-308}$.

treat each base-8 digit as an independent value and rewrite each base-8 digit as the corresponding 3 digits in base-2 and then combine them end-to-end.

Since $16 = 2^4$, to convert base-2 to base-16, instead group the digits of the binary value in groups of 4 (adding high-order zeros to get a multiple of 4 digits). The same binary value becomes (0001)(1111)(1110), treating each group as a separate value gets (1)(15)(14), converting each value to the corresponding valued symbol in base-16 gets (1)(F)(E), and $1FE_{16}$ is indeed the representation of the same value in base 16. To reverse the process simply treat each base-16 digit as an independent value and rewrite each base-16 digit as the corresponding 4 digits in base-2, and combine end-to-end.

Unless an integer value is so large that it exceeds the maximum number of digits that can be stored, any integer value can be exactly represented in any base. Computer numeric representations limit the maximum number of digits, so when that limit is exceeded either the number cannot be stored or an approximate value with a limited number of significant digits is stored. The user should receive an error indication if a value cannot be represented.

Non-Integer Values

Values from measurements in the real world rarely have exact integer values. Depending on measurement tools used, real world measurements are either made to a certain number of decimal places or precision, or perhaps to a nearest fractional subdivision, like a length measurement to the nearest 1/16 of an inch. While students in grade school are taught how to do basic arithmetic of addition and multiplication by hand using fractions, and calculations with fractions that leave results in fractional form can always be exact, keeping all intermediate and final results in the form of fractions for complex real-world calculations quickly produces results with very large numerators and denominators that are impractical to use and which may imply a precision that is not warranted.

For more complex calculations, one is taught to instead use decimal numbers with a fractional part separated by a decimal point. The rules for performing calculations with values expressed to a given number of decimal places, followed by rounding results to a precision appropriate to the precision of the original values, are both easier to perform and make it clearer that the accuracy of an answer is limited by the accuracy of the original data.

With values expressed in decimal (base-10), the value represented by digits to the right of the decimal point have progressively smaller weights, 1/10, 1/100, 1/1000,… etc progressing further to the right. From our familiarity with working with decimal fractions, we intuitively understand that a value with three decimal places can represent a value to the nearest 1/1000th, and even exactly represent any value that happens to fall exactly on a 1/1000th boundary.

When a computer represents a value in base-2 with a fractional part, the values represented by digits to the right of the binary point have progressively smaller weights also, in this case ½, ¼, 1/8, 1/16, etc. So it you have a binary fraction with 10 binary places to the right of the binary point, you can express a value rounded to the nearest multiple of $1/2^{10}$ or 1/1024, roughly equivalent to being able to express a value to the nearest multiple of 0.000976562 in base 10.

While it is true that one significant digit in a decimal representation roughly corresponds to about 3.32 significant digits in a binary representation (the ratio of log 10 to log 2), what is not intuitively obvious is that fractional

values that may be exactly represented in one base may require many more places than that ~3.32 factor would imply to exactly represent the same value in a different base. We also find that some very simple fractional values, like 1/3 cannot be exactly represented as a decimal fraction. It can be approximated to any desired degree of accuracy (0.3, 0.33…, 0.3333, etc.), but can never be exactly represented. What is not intuitively obvious is that although a 10 bit binary fraction can express values to a slightly higher precision than a 3-digit decimal fraction, not only can a very simple decimal fraction like 0.1_{10} not be represented exactly by a 10 bit binary fraction, it can't be exactly represented by any binary fraction. The fraction 0.1_{10} becomes an infinitely repeating binary fraction ($0.000110011(0011)..._2$) that cannot be exactly represented in binary by any number of bits.

Programming languages and user interfaces to spreadsheet applications typically allow the user to specify fractions in base-10 for values that are actually stored in base-2. That means that fractions that the user considers to be exact, like 0.1, may actually be stored as an inexact approximated value. This can cause unexpected results in some cases. For example, using Excel to add various powers of 10 to 0.1 and then subtracting that same power of 10 value off again and displaying the result to 12 decimal places gives the expected result of 0.100000000000 up to 10^4, but for 10^5 and above strange garbage starts to appear to the right:
When 10^{14} is reached, adding 0.1 just gets an intermediate result of 10^{14} and a final result of 0.0 .

100000	0.100000000006
1000000	0.099999999977
10000000	0.099999999627
1E+08	0.099999994040
1E+09	0.100000023842
1E+10	0.100000381470
1E+11	0.100006103516
1E+12	0.099975585938
1E+13	0.099609375000
1E+14	0.000000000000

This is not the behavior one would expect if the calculations were being done in base-10. In base-10 one would expect to consistently get back the exact same 0.1 value until the power of 10 reached the number of significant digits that could be handled, causing the 0.1 to be dropped when the larger value was added with a 0.0 final result. By doing other tests it can be shown the "garbaged" result values obtained for 10^5 through 10^{13} are consistent with loss of bits in base-2 arithmetic from the approximate binary representation of 0.1. Both Excel and Calc do a good job of hiding this behavior by rounding displayed results. If the results above are displayed to only two decimal places, all results are consistently displayed after rounding as "0.10" until changing to "0.00" on the last line. This is what one would expect if the calculations were done in decimal, but in this case the displayed values would be deceptive: If you were to compare the last two "0.10" displayed result values, we know from displaying the values with more decimal places that you would find they are not equal.

One can experiment and find other cases where the lack of an exact representation of 0.1 in base 2 causes strangeness:
=100+0.1-100 with the cell formatted for 15 places yields 0.099999999999994 instead of 0.1;
adding 522 cells containing 0.1, formatted for 13 places yields 52.1999999999999 instead of 52.2 (a sum of fewer than 522 cells displays sums that appears exact to the nearest 1/10)

Conclusions

Unless a computer application or the implementation of a programming language explicitly says that calculations will be done in base-10, you should assume that they are done using more efficient computations in base-2, with all the limitations that implies on representing some simple decimal fractions exactly. It is technically possible even on a binary-based computer to create a computer application that will store numeric values in base-10 and do all arithmetic calculations using base-10 arithmetic; but this is rarely done because it is more complex to do so

and would have noticeably poorer performance than using the base-2 functions directly supported by the computer hardware.

Do not assume that fractional values entered or displayed are exactly stored, even though they may be displayed as exact; or that two displayed fractional values that are displayed identically are indeed identical. Any spreadsheet logic or program logic working with numeric values based on numbers with fractional parts needs to take this into account and be aware that comparing values derived from fractional values for exact equality may not work as expected. Where this becomes a problem, It may be possible to use the ROUND function to eliminate non-significant differences between two values with fractional parts.

These caveats can also apply to spreadsheets containing US currency values expressed to the nearest cent, as 1/100 like 1/10 lacks an exact representation in binary. Because of the number of significant digits retained and the rounding of displayed results, using spreadsheets for personal accounting or for most business applications you shouldn't see any problems. But, if you you try to track the U.S. national debt to the exact penny using a spreadsheet, that is getting close to the point where entered values and computed results might not have enough significant digits in Excel or Calc to resolve correctly to the nearest penny, and values could easily get off by a penny or so.

Understand the limitations when a computer application uses base-2 representations internally.