Mit nevez(z)ünk pontosságnak?

Sokan nekem szegezték már a kérdést: meddig lehet bízni a számítógép pontosságában? Erre vonatkozólag magam is számos választ hallottam már, melyek többsége az adott helyzetben, az adott feladatot nézve meg is állta a helyét. Érdemes a kérdés által felvetett problémakörben kissé mélyebbre ásni, hiszen a csillagászat azon kevés tudományok egyike, ahol extrém pontossági követelményeknek kell eleget tenni. Itt a más területeken még megengedhető kerekítések, elhanyagolások legtöbbször olyan mértékű hibát visznek a számításokba, melyek a végeredményt nem csupán kétes értékűvé, de teljességgel használhatatlanná tehetik – és teszik is.

Az ember első lelkesedésében gyakran esik abba a hibába, hogy a számítógép által kezelt számokat, a végzett műveleteket abszolút pontosságúnak tekinti. A számítás végeredményét pedig – mint égi kinyilatkoztatást – az utolsó kiírható jegyig felülbírálhatatlanként kezeli. Ez sajnos korántsem ilyen egyszerű...

Lássuk az okokat:

A gépi számolás pontatlanságáért két „mágikus" fogalom felelős: a *számábrázolás* és az *aritmetika*. Igazán a két dolgot nem is lehet (és szabad) ennél jobban szétválasztani, hiszen egymás nélkül értelmezésük értelmetlen, de a legjobb indulattal is felesleges.

Mit takar e két „bűvös" szó? Aki hallott már valamennyit a számítógépekről (és van-e manapság, aki nem?), együtt kezeli róluk alkotott képét egy jelzővel, tudniillik hogy *napjaink számítógépei digitálisak*. Ez pedig érthetőbbre fordítva annyit tesz, hogy a gép az általa kezelt bármely mennyiséget az alapegység megszámlálható többszöröseként, egy szám alakjában (azaz digitálisan) megjelenítve ábrázolja. Ennek a megjelenítésnek a módja (hiszen igencsak sokféle lehetőség adódik) a számábrázolás.

Már a számítástechnika hőskorában (amely – ne felejtsük el – csupán néhány évtizeddel ezelőtt volt!) kialakultak a számok ábrázolásának legkézenfekvőbbnek látszó formái. A célszerűség első ránézésre nem is olyan nyilvánvaló. Megértésükhöz még egy fogalom tisztázása szükséges: mégpedig hogy a számítógépek alapvetően a *bináris* számábrázolási módokat teszik lehetővé. Ez alatt pedig azt értjük, hogy a gép számára kezelhető bármely adatnak kettes (azaz bináris) számrendszerben felírhatónak, vagyis nullák és egyesek sorozatából állónak kell lenni. Mint minden számítástechnikával kapcsolatos általános kijelentés, ez sem abszolút értelemben igaz, csupán azt jelenti, hogy a gép fizikai felépítésénél alkalmazott kétállapotú elemek természetéből adódóan ez a legáltalánosabb számábrázolási mód. Nem tévednek tehát azok sem, akik egyes processzorokkal kapcsolatban az *oktális* (nyolcas), a *hexadecimális* (tizenhatos) vagy netalán a *decimális* (tízes) számrendszert emlegetik. A nyolcas és tizenhatos számrendszer könnyen kapcsolható a ketteshez, hiszen mindössze hármas illetve négyes csoportokba kell a bináris jegyeket osztanunk, mely csoportok megfelelnek az adott számrendszer egy-egy helyiértékének. A tízes számrendszer használatát egyedül az ember által teremtett hagyomány indokolja. Az így megjelenített adatokat feldolgozás, tárolás előtt általában át kell alakítani, majd a könnyebb érthetőség kedvéért a műveletsor végén vissza kell alakítani a köznapi értelmezés számára is „emészthető" formára.

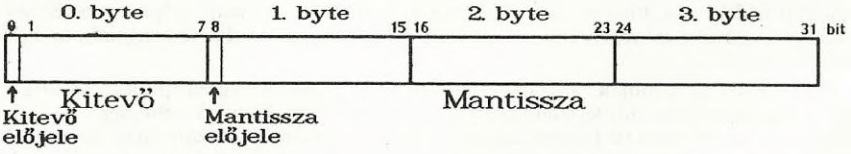
Hogy példát is mondjak: a százhetvenkilenc, mely mennyiség elképzelése és megértése még a leggyakorlatlanabb fejszámolóknak sem okoz leküzdhetetlen nehézséget, a gép számára csak az 10110011**b** bináris számként jelenthet valamit (a szám után illesztett **b**, **o**, **h** vagy **d** a szám bináris, oktális, hexadecimális illetve decimális voltára utal), melynek hétköznapi visszafejtése: 1*128+0*64+1*32+1*16+0*8+0*4+1*2+1*1 = 179**d**. Csak az elvetemült bitvadászok ismerik fel azonnal a B3**h** hexadecimális, vagy a 263**o** oktális számban

ugyanezt az értéket. (Akit a számrendszerek mélyebben foglalkoztatnak, bármely összefoglaló matematikai zsebkönyvből kielégítő mennyiségű információhoz juthat.)

De kanyarodjunk vissza az eredeti gondolathoz! Mi köze van az ábrázolásnak a pontossághoz? Hiszen a bináris 10110011**b** éppoly pontosan ugyanazt az értéket jelenti, mint a 179**d** vagy a B3**h**. Ez így is van, ám ne felejtsük el, hogy nem csak pozitív, egész és nem különösebben nagy vagy kicsi számokból áll a világ. A számítógépek egyik legfontosabb (s manapság talán legdrágább) alkateleme a *memória*. Az a tárterület, ahol a gép – bináris számok formájában – a prógramo(ka)t és a számítások adatait raktározza. Bár a mai memóriák akár milliószor nagyobbak is lehetnek, mint néhány évvel ezelőtti eleik, mégsincs akkora kapacitásuk, hogy a tárolóhely korlátlanul álljon rendelkezésre. Így meg kellett szabni a határokat, mekkora területet foglalhat el egy-egy számadat. Itt kerülnek elő a számítástechnikában megszokott mennyiségek: a *bit* (elemi tárolóegység, értéke egy bináris jegy, azaz 1 vagy 0 lehet), a *byte* (a magyar helyesírás szabályai szerint: bájt, mely némileg önkényesen, ám a gépek fizikai adottságaihoz igazodva – általában – 8, egységként kezelhető bit), a *word* (szó = két, együtt kezelt byte), stb...

Lássuk csak, mire elegendőek ezek az egységek! A legegyszerűbb egység a byte, mely 8 bit lévén 0-tól 255-ig alkalmas a számok megjelenítésére. Előző példánkban ekkora tárterület tökéletesen elegendő is volt. Ha ennél nagyobb számot kívánunk ábrázolni, kézenfekvő a következő egység, a word használata, mely 2 byte, azaz 16 bit, így már 0-tól 65535-ig képes számokat tárolni (Mennyivel szebb ez utóbbi a hexadecimális FFFF**h**-ként felírva!). Sokkal tovább nem folytatható a sor, mert a mai hétköznapokban olyan gépek terjedtek még csak el, melyek fizikai ábrázolásmódja maximum 32 bit szélességű számokat engedélyez (ilyenek pl. a 386-os PC-k). Ez pedig FFFFFFFF**h**, azaz, ha jól számolom, 4294967295**d**. Hol van még ez az alig 4,3 milliárd a (mindkét értelemben) csillagászati számokhoz, melyeket kezelni kell!... Mi történik viszont abban a sokkal egyszerűbb esetben, ha a -300-at próbáljuk gépünk szilícium agyába erőltetni? A megoldást az előjel bit bevezetése jelenti, azaz a fent leírt bináris számok legnagyobb helyiértékű bitjét nem mint számot, hanem mint előjelet értelmezzük. Ha ez a jegy 1, a szám negatív, ha pedig 0, akkor pozitív. Frappáns megoldás, ám értékéből levon, hogy pl. a harminckét bites számot tekintve az előjel helyiértéke már 2147483648, azaz ennyivel kisebb a legnagyobb felírható szám. Pedig még nem is próbálkoztunk a tört számok ábrázolásával! A tízes számrendszerben megszokott tizedespont mintájára természetesen mód van kettedes, nyolcados vagy tizenhatodos pont bevezetésére, de ez megint több technikai problémát vet fel. A legfontosabb mind közül, hogy hol legyen a törtpont. Nem egyszerű kielégítő megoldást adnunk, ha a (mint láttuk) fix hosszúságú számok jegyei közé kell a törtpontot kitennünk, mégpedig oly módon, hogy a borzasztóan nagy és az elképzelhetetlenül kicsiny értékek is leírhatók maradhassanak.

Két általánosan elterjedt módszer született: a fix- és a lebegőpontos számábrázolás. Az első alig szorul magyarázatra. A törtpont (a megtárgyaltakból adódóan: kettedes pont) rögzített helyen van, tőle „jobbra" a tört rész, míg „balra" az egész rész áll. A fixpontos számábrázolás hallatlan előnye, hogy ezeket a gép közvetlenül kezelni tudja. Az igazán forradalmian újat a lebegőpontos ábrázolás hozta. Itt a számot a tudományos gyakorlatnak megfelelően két részre bontjuk. Az egyik rész a mantissza, a másik a kitevő. Tízes számrendszerben ez a megszokott 1321,4 = 0,13214*10^4 alak, ahol az 0,13214 a mantissza, a 4 pedig a kitevő. A teljes ábrázolt adathossz (a mantissza előjele, a kitevő és a mantissza) 4 vagy 8 byte szokott lenni egyszeres illetve dupla pontosság esetén, a következőképp felosztva:



Ha pl. a kitevő számára 8 bitet tartunk fel, akkor az előjel biten kívül még 7 bit marad a kitevő értékére. Ez pedig a dolgokat egyszerűsítve azt jelenti, hogy a legkisebb ábrázolható

szám 10^{-127}, míg a legnagyobb 10^{127} nagyságrendű. Nézzünk csak meg egy-egy ilyen nagyságú (vagy kicsinységű) számot:

10000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

vagy

0,0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001

Bizalomgerjesztő, nemde? Van értelme még mindig pontosságról elmélkedni, amikor ilyen észveszejtően parányi vagy gigantikus mennyiségeket is kezelhetünk gépünkkel? De mennyire! Nem tévesztendő össze ugyanis egy szám *nagyságrendje* a *pontosságával*. Eddig a kitevő szerepét mutattam be. Most jöjjön a pontosság szempontjából sokkal érdekesebb mantissza. Az imént, mondjuk dupla pontosságú (azaz a gép pontossági határait már-már súroló) számunk 8 byte-jából egyet elhasználtunk a kitevőre. A maradék hétszer nyolc bit tehát a mantissza. Ha az előjelét is számítjuk, marad 55 bit az értékes jegyekre. Tízes számrendszerben megfogalmazva nagyjából 16 értékes számjegyet jelent csupán. Ez bizony mesze elmarad a fentebb leírt – ám ábrázolható – 128 jegyű számoktól.

Általánosságban elmondható: a PC kategóriájú számítógépeknél a különösebb bűvészkedés nélkül elérhető pontosság 11-12 értékes jegyet jelent. Matematikai segédprocesszor (co-processor) használatával ez nehézség nélkül feltornázható 15-16 jegyre. Ha pedig különlegesen pontos számításokat kívánunk végezni, néhány programozási nyelv felajánl bizonyos kiterjesztett pontosságú számtípusokat, melyek 19-20 tizedes jegyig pontosnak értelmezhetők.

Nézzünk egy egyszerű, hétköznapi példát! Szinte nem akad komolyabb csillagászati számítás, ahol ne kellene Julián dátumban adott időkkel manipulálni. Hogyan is van ez? A Julián dátum egész része is tekintélyt parancsoló milliós nagyságrendű. A pontosságot a törtnap ábrázolt jegyeinek száma adja. Meddig elegendő a gép számábrázolása ehhez az időadathoz? Tegyünk egy próbát. Mondjuk tizedmásodperc pontossággal figyeltük meg valamely esemény bekövetkeztét. Az időpont legyen 1993. január 1. 12:00:00.7 UT. Ez átszámolva Julián dátumra: 2448989.0000081018518 5179. Írjuk most fel ezt a számot a felsorolt három pontossági osztálynak megfelelően:

12 jegyre: 2448989.00000 = 1993.01.01. 12:00:00.00

16 jegyre: 2448989.000008101 = 1993.01.01. 12:00:00.6999264

20 jegyre: 2448989.0000081018518 = 1993.01.01. 12:00:00.699999995

Tehát: hiába képes a gép pontosan ábrázolni szinte tetszőlegesen kicsi vagy nagy számokat, ha a két dolog találkozik (nagy számoknak kicsiny törtrészeire is szükség van), kezd a dolog bizonytalanná válni. Ha a hobby szinten még mindig legelterjedtebb 8 bites gépekeket (Commodore, Sinclair, HT, Primo stb...) tekintjük, ennél sokkal siralmasabb a kép. Az elvesző tized- vagy századmásodpercek nem tűnnek komoly veszteségnek, de ha a sok ezer, tízezer művelet során mindig csak egy-egy századot téved a gép, a végeredmény eltérése a valóstól már tetemes... Jelen példa már csak azért is sántít, mivel kitalálták a módosított JD-t, melynek használatával több értékes jegyet spórolhatunk, a számábrázolás pontosságra való hatását azonban e példa is kellőképp megvilágíthatja.

Ezzel elérkeztünk a másik említett fogalomhoz: az aritmetikához, mely egyszerűen fogalmazva azon módszerek összessége, ahogyan, amelyekkel a gép az adott módon ábrázolt számokkal a műveleteket végzi. A kutya legtöbbször itt van elásva. Amint az iménti JD–MJD összevetés is szemlélteti, egyszerű trükkökkel is túl lehet lépni a számábrázolás által emelt korlátokon. Az aritmetikába viszont közönséges halandónak nem nagyon van beleszólása, ez a gép legbensőbb magánügyei közé tartozik. Igaz, létezhetnek olyan, különleges feladatok, amikor nincs más lehetőség, aritmetikát kell írni, mondjuk 80 bitre, de – bár magam is tettem már ilyet – erről legmélyebb jóakaratom jeléül ezennel mindenkit lebeszélek.

Az aritmetikáról nem kívánok, az igazat megvallva nem is nagyon tudok hosszabban értekezni. Itt a hibák halmozódása a legnagyobb veszélyforrás. Arról van ugyanis szó, hogy nem csak a végeredmény terhelt hibákkal, hanem az összes köztes eredmény is, sőt, nemegyszer a kiindulási adatok is. Az egyes műveletek során pedig (ne feledjük: nem egy-kettő, hanem akár több millió elemi számításról van szó, melyeknél az eredményeket a gép csonkíthatja, kerekítheti) a hibák nem csak összeadódhatnak, de szerencsétlen esetben akár hatványozódhatnak is. Röviden összefoglalva: ez a dolog volt oka eddigi legtöbb

bosszúságomnak, ha nagyobb pontosságú számításokra kellett programot terveznem. Bizonyára többen meglepődtek már, mikor is hőn szeretett C-64-esük a gyök 9-et négyzetre emelve valami egészen furcsát, de a fejben is gyorsan kiszámolható eredménnyel csak nagy vonalakban megegyezőt írt ki a képernyőre. Ez bizony durva hiba. Kevesebben vannak már azok, akik a PC-n futó Turbo Pascal fordító lebegőpontos kerekítésére panaszkodnak; pedig ez ráadásul periodikus hibát visz a számításokba. Aki eddig elolvasta irományomat, és találkozott már irreálisan és látszólag ok nélkül rossz számítási eredményekkel, sugalmazás nélkül is ismeri a következtetést: a gép által szolgáltatott eredményeket bizalommal, ám kellő körültekintéssel kell elfogadnunk, szem előtt tartva a lehetőségeket. Ezek a – főként a hardware felépítéséből adódó – korlátok odafigyeléssel és (ne feledjük el!) kellő szaktudással nagyrészt leküzdhetők. A fontos, hogy tudjunk róluk!

Végezetül hadd ismertessem egy teszt eredményét, melyet munkahelyemen, a Kozmikus Geodéziai Obszervatóriumban végeztem annak illusztrálására, hogy egy feladat bármely (!!!) számítástechnikai eszközzel megvalósítható. Ugyanakkor a megoldás előkészítésénél mérlegelni kell a követelményeket és lehetőségeket, ugyanis a különböző eszközök (itt: programozási nyelvek, fordító programok) más-más hatékonysággal segítik a programozót a végeredmény elérésében.

A teszt-feladat egy egyszerű, csillagászati alapfüggvény, a módosított Julián dátum kiszámítása volt. Az algoritmus alapja a Hatcher-féle eljárás. A teszt végeredménye a 20. század minden napjának 0 óra UT-re vonatkozó módosított Julián dátuma, melyet a program Gregorián dátumból számít. A megoldás 11 fordítóra, 7 programnyelven született meg, s eredménye a következő:

A tesztben csak azon programok vehettek részt, melyek minden eredménye a többivel azonos pontosságú, és kizárólag software aritmetikát használtak (azaz nem co-processort).

COLUMNS(12), DIMENSION(IN), COLWIDTHS(.54,.50,.57,.58,.56,.50,.45,.51,.48,.60,.39,E1), HGUTTER(.056), VGUTTER(.056), BOX(Z_DUPLA), HGRID(Z_EGYES), VGRID(Z_EGYES), KEEP(ON), RULE(Z_EGYES,R0C0..R1C1), RULE(Z_DUPLA,R1C0..R1C11), RULE(Z_DUPLA,R1C11..R1C12), RULE(Z_DUPLA,R0C1..R5C1), RULE(Z_REJTETT,R0C0..R0C1), RULE(Z_REJTETT,R0C0..R1C0)

TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG, TÁBLA SZÖVEG

, MS Fortran v5.1, Desmet C v2.4, Borland C++ v3.1, Borland Pascal v7.0, Turbo Basic v1.1, Quick Basic v4.5, Quick C v2.5, Turbo Pascal v5.5, TopSpeed Modula v2.0, Fast, MASM assembly v6.0

Kiírva (sec), 330, 366, 261, 258, 440, 440, 266, 262, 283, 295, 271

Csak számolva (sec), 2, 3, 1, 1, 8, 9, 2, 5, 2, 11, 0.5

Fileba írva (sec), 55, 458, 25, 14, 440, 440, 18, 17, 178, 439, 393

Méret (Byte), 27424, 7680, 16449, 16784, 33747, 33744, 31549, 18510, 16440, 2253, 621

Az eredményeket szóban is összegezve:

Az összes program ugyanazt szolgáltatta, más-más áron. A táblázatból ez nem derül ki, de a legnehezebben a két BASIC fordítóból lehetett helyes eredményeket kipréselni. Általában azonos végadatokat produkáltak a többi programmal, ám *néha* tévesztettek, mégpedig elég nehezen felderíthető módon. (Pl. két különböző naphoz ugyanazt a Julián dátumot rendelték.) Az ilyen „néha" a programozók nem csekély hányadának okoz idő előtt ősz hajszálakat. Talán mondanom sem kell, hogy a legnagyobb galibát a BASIC szabad típuskezelése (számábrázolás!) okozta, mivel a fordító hol ilyen, hol olyan módon ábrázoltnak (egész, lebegőpontos, dupla pontos...) tételezte fel az adatokat – minden jogalap nélkül. Többszöri rákényszerített típuskonverzióval sikerült rakoncátlankodását megfékezni.

Tanulság: ha nem is lehetetlen, nagy pontosságú, hosszabb lélegzetű programot csak önsanyargatásra hajlamos és hosszabb klinikai kezelésre felkészült emberek írjanak vagy írassanak BASIC-ben. Ha pedig mégis erre szánja valaki magát, törődjön bele a szinte elkerülhetetlen kudarcba. Természetesen ezernyi, amatőr célokat teljességgel kielégítő program írható e könnyen elsajátítható és sokak által ismert nyelven, de ismerek olyan alkalmazásokat is, ahol „hozzáértők" pénzt és időt nem kímélve, BASIC-ben kívánnak tudományos célú pályaszámításokat végezni... *Ne tegyük!!!*

A táblázat utolsó két oszlopa is érdekes lehet. Valószínűleg kevesek által ismert az itt jelölt FAST fordító nyelve. Ez egy PC-re készült, szabadon terjeszthető programnyelv (a SolarSoft-nál bárki megvásárolhatja kb. 500 Ft-ért). Nevéhez illően (fast=gyors) apró és gyors programok készítésére alkalmas nyelv ez. Kis szépséghibája, hogy számábrázolásában a csúcsot a tizenhat bites egészek képviselik. Itt és az utolsó oszlopban szereplő assemblyben – ha a programozó ragaszkodik a nyelvhez – kénytelen-kelletlen át kell venni a magasabb szintű nyelvek fordítói vagy a co-processor szerepét, és megvalósítani a 16, 32 vagy magasabb bitszámú aritmetika műveleteit. Ha a programok hosszát nézzük – megérte az erőfeszítés. Ha a befektetett munkát tekintjük – már kétségek adódnak.

Tanulság: mindig az adott feladathoz leginkább illő programnyelvet használjuk. (Tudom: jó annak, aki többet is ismer...) Ha a teszt-feladat egyik kitétele az lett volna, hogy a számítás nem foglalhat 1 kByte-nál nagyobb helyet, nem esik gondolkozóba az ember – assemblyben dolgozik. De ha adott a választás valódi lehetősége, hát éljünk vele! Tisztelem a hagyományokhoz hű, Fortranban programozgató tudományos köröket, a gépi kódban virtuózkodó ifjú programozókat, vagy a Clipperben MegaByte-nyi programkódot generáló adatbázis kezelőket, s magam is gyakran nyúlok hol egyikük, hol másikuk eszköztárához. Mégis: korszerű és hatékony általános programozási nyelvként ma leginkább a Pascalt és a C-t ajánlhatom úgy a PC, mint a kisebb kategóriájú gépeken. Indokokért elegendő a fenti táblázatot kicsit áttanulmányozni.

Végül pedig: tudom, az amatőrök túlnyomó hányada kis teljesítményű számítógéppel, otthon dolgozik; BASIC-ben. Nem ellenük, hanem segítségükre soroltam fel néhány tapasztalatomat (Ne feledjük: a tesztben a BASIC programok is ugyanazt az eredményt szolgáltatták, mint agyondicsért társaik!). Nem csak luxuskocsival, de lóháton, sőt gyalog is be lehet járni nagy távolságokat, és sokszor a hátizsákos turistát több, szebb élmény éri, mint a sötétített üveg mögött, légkondicionált utastérben, az autópályán hihetetlen sebességgel száguldó meseautó utasait...

HEITLER GÁBOR