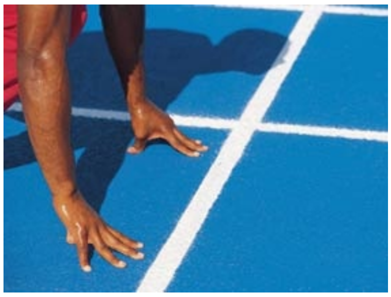
maXbox



maXbox Starter 41

Deal with Big Numbers

1.1 A Big Decimal or Big Int Interface

Today we step through numbers and infinity.

As you may know there's no simple solution to print, calculate or store big numbers or decimals, for example you want to compute 400000078669 / 2000123 your calculator shows (so does my Casio FX-880P):

199987.7401

So this is not the end of the line, a second test is

maxcalcF('400000078669 / 2000123')

and we get: 199987.740088485

And there are even more numbers that need to compute so we switch to http://www.wolframalpha.com to get the real precision thing or at least an approximation:

199987.7400884845581996707202507045816682274040146530988344
21683066491410778237138415987416773868407092963782727362267
220...

http://www.wolframalpha.com/input/?i=400000078669%2F2000123

again as you suppose the numbers go on.

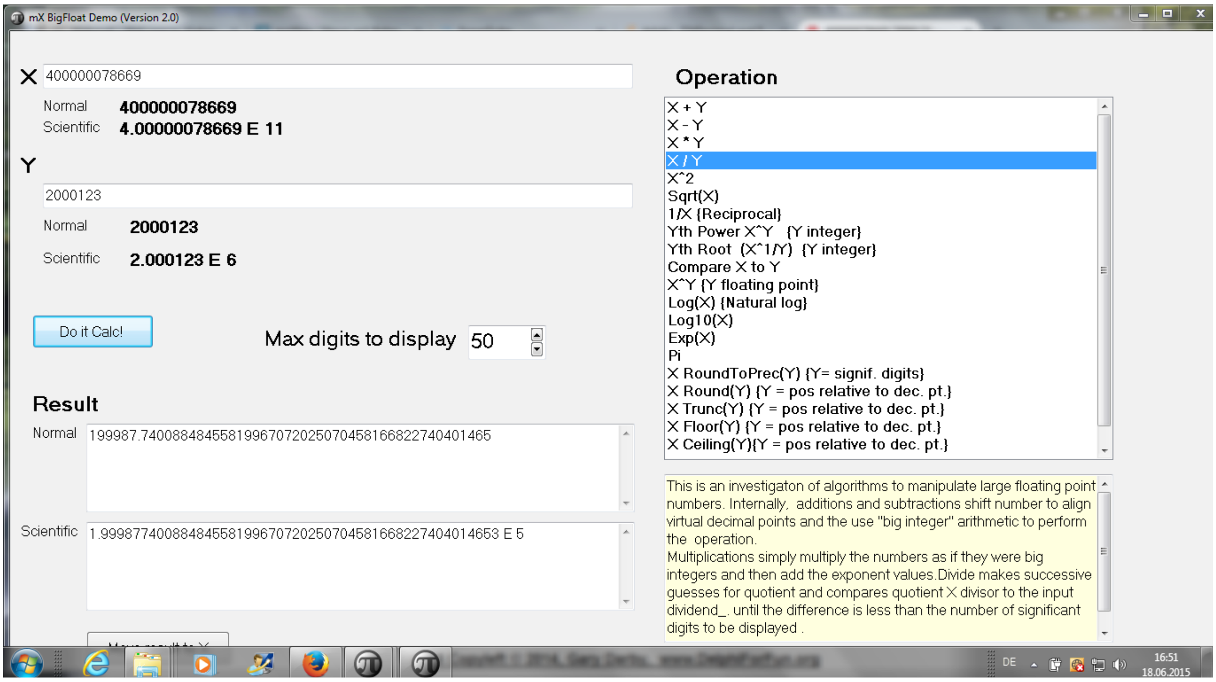
Use "Power Towers" to write them down. The decimal point is the most important part of a decimal number like above. Without it, we would be lost ... and not know what each position meant.
Dividing decimals is almost the same as dividing whole numbers, except you use the position of the decimal point in the dividend to determine the decimal places in the result. Our division is always an approximation.

Approximate means you're going to round the number. Because you're not actually giving the exact number, all those numbers after the decimal, the rounded number is called an approximation:

199987.7401 is roundToPrec4 of: 199987.740088485

Although, you probably wondered how they get those nice and fancy graphical user interfaces (GUI) for large numbers, here in maXbox we do also have one:

maXbox3 `568_U_BigFloatTestscript2.pas` Compiled done: 6/18/2015



The idea that you are approximating is that, as you are only taking the first 50 decimal places as you can see at the screen-shot.
The same like wolfram goes like this:

199987.7400884845581996707202507045816682274040146530988344
2168306649141077823713841598741677386840709296378272736226 7
2205

When we try to write this decimal number (or the well known PI or SQR(2)) in decimal notation, we get an endless stream of digits.
3.141592653589723.....and so on forever.
But suppose instead, we use fractional notation. Then we can write each part as a precise (irreducible)

$$\frac{400000078669}{2000123}$$

A fraction is an exact ratio of 2 numbers, and if those 2 numbers are integers, or at least rational numbers, then the fraction can more

appropriately be called a rational number. An irrational number can be represented as an approximation to a rational number to an extremely high degree of accuracy.
It's quite clear that there are fractions which can't be expressed in finite decimal form!
Now, here's the big problem. Not every number is rational! For example there is no fraction for sqrt(2). That is, no matter what whole numbers m and n you pick, m/n is not the square root of 2. Euclid wrote down a real AND beautiful proof of this fact around 2300 years ago.

Interesting point about those real numbers is also the possibility to divide the number to his prime factorization:

29×37×127^(-1)×179×15749^(-1)×2082607

```
maxcalcF('29*37*(127^-1)*179*(15749^-1)*2082607');
```

>> 199987.740088485

1.2 Real Big Integer

So what about big integers? For example you want to compute fact(70), your calculator shows:

fact(70) = 1.19785716699699e+100 or **maxcalcF**('70!')

1.19785716699699E100

or even more

1.19785716699698917960727837216890987364589381425464258 57... × 10^100

but the maximum range on Pascal, C or Delphi depends on your operating system types, means nowadays an int64 range is big.
Now that the "signed" words are finally up-to-par with the unsigned integer types, languages introduce a new 64-bits integer type, called Int64, with a whopping range of -2^63..2^63 - 1

Another way is to use a type extended, but the limitation is precision like

```
Writeln(FloatToStr(Fact(70)))
```

it only shows 1.2E+0100 or 1.19785716699698966E100

With a BigInt Library you'll see the full range of Fact(70):

1197857166996989179607278372198789275553662800958278984531968000000000000000

All examples can be found online:

maxbox3\examples\161_bigint_class_maxprove2.txt

http://www.softwareschule.ch/examples/161_bigint_class_maxprove2.txt

The call respectively the calculation goes like this:

```
function GetBigIntFact(aval: byte): string;
//call of unit mybigint
var mbRes: TMyBigInt;
   i: integer;
begin
  mbRes:= TMyBigInt.Create(1);
  try
    //multiplication of factor
    for i:= 1  to aval do
      mbRes.Multiply1(mbres, i);
    Result:= mbRes.ToString;
  finally
    //FreeAndNil(mbResult);
    mbRes.Free;
  end;
end;
```

Or you want the power of 100 like 2^100=
12676506002282299670376

```
function BigPow(aone, atwo: integer): string;
var tbig1, tbig2: TInteger;
begin
  tbig1:= TInteger.create(aone);
  //tbig2:= TInteger.create(10);
  try
    tbig1.pow(atwo);
  finally
    result:= tbig1.toString(false);
    tbig1.Free;
  end;
end;
```

At least one really big, it's 333^4096 (10332 decimal digits)!:

85424105895770887322966965917914584710138161386222147182917
67781049536057906627318361093758865620577697322240787369539
81504332246815140327668478948527046875787550310970504170251

82159123158149832522632506558096888465749900859669714028055
7172226167217308161576577272999913389932448720516828003070
67049948583875478861163177472370389199969581395362347697256
70960895462872160261732690973238984844130761428601643232816
96899530174488741933479651249141554621396684104620616204908
08738502674650541288448161671070156327238251274328179200400
93890996761080835353394335725524764503872061091991652109449
19392226034862115885018213075725104899320094825429972785833
08567702799428942416066175248810727171285940075837283674910
40238020148038165962562077292113243073436214316216978130335
33870396651279893801306533507055182475982690042018637954924
71837910714543445430478178723995293717173465603902499439767
52390512439014099231872228674262774255372493330431055176825
66636232366673489784851223121246465215815916502622714756503
41470134146173462494457636856525872971987877831270158832603
81720501132263714345569154398039964038547327865662026993558
29785045595252413682840639314284074867518100746598657655945
81378019251453464202570850546505546651918600042626257608188
49535769666297091453700207198739748884149693263644965541429
60625942272943281855130658659637414117763954618293009737119
20497440723531587803915956018581696810199146742826127160616
96675183761740650324877602818378317677730431971470829242037
19559841427945394611344759210789672710312151288626779198995
38701143923451064706611017647623102481237637914918894554224
08190814870673307938473049085663216256407994368677685271087
59041690965302269272779289181709898614618400817701817670333
89755570766416722785651840954388568904657279312763515446538
14727117240451212404891260149108022108352756826060916165752
75842968506140334676934667568824407601228835743948301133856
67291848920157649283141024291690280652494545083790886477617
43720126167236379124481127893994316258310107716760682781353
13052205927671502890929477717785458301559276426947626959261
74086054203450601228998156862816770206156286995619547713403
66063324712132695944068411905946566611734854030824623340469
92360387210154102113393219475674943157388839887567715636613
34205014438776231018691956945896654929451766257449857738007
27444065544750842262982606864122383198213933891466795134120
80961172228352857918693293077782844810890632478086472100076
88704100771174997144137624681757514590807817999921433168370
14153195459799923678729819655105346515098491418037313214801
87994144154444550442819531303936818323269177814053969040746
51701879257716143646994952012988801927312646343698172158367
61912526640574323621879838059898890627520605936959551652405
90809647635413089058367300817220960358021625376219618474184
69992177098710584107130852431474071403333632860325415187194
48162274704214637381912929168658838072372439957470569861598
51679911341687043236694397948103449066652714018257382920338
12042010509830956293769066454597716029454679602570172933558
47426310376543216436257655693564688938168477374639916971824
02728135514271928199771524104739484573797062834940055581577
67229410238117423497658874466085698219698093781073032946381

54093923887380662374997946632996174973155219412977143689536
93626346659433245937550709818320269028376442759671548311150
29681760031234141791574578094471517422412689480662593361694
92504435874599155310915433868466432350489066111107816884909
61835339066524922842142508725963101026026236290675930196731
73552799026653846151496995266112396254547674003723688793668
70430969934783220357570165871596112617519081177716288358775
11692756033606840409034957153520370203025944206641163576863
92424070432686477410458637592771813119302505708045545791331
74227674636089201829445749466630340510465170537923125432328
57839740121048013161681915430491815656845135427468061756390
04934440540270331262335589548473966818981111982651233800107
48747405008943837472732619075280509937188130644867401433861
57681010034027203393276895085751311506894111033284657262265
58155966132471197425573248051246364064100968284789300972872
58388209492425648666245060219257890753714689218831797215716
60390788734493878267923484583108882977346724409885092951438
78217708481814130975623645815400519531072329972316424125029
83311338676548007573475077494264378039776424697719572306930
88766695667158451604134516400656364381872715873008395723257
97940380735398040659721222058817651087721809880276579722510
39708526658847167714129120531620805313149564483218372281419
53368247624399223724601214570091865759014802662737858014014
79321896896482250699296283841144875763989136349504660143195
13033816141307450057013723937604815534045838161179374965402
69262035790501488443161926140136755446024936592443075520866
73181833019464091484685855011484242655933958222567738947458
12844034848259768590120326004103399895146100148171902711195
78683839026679616530698635588982885464930887373647147964139
90901429121521451261576282787526544172701709313410194780946
90999402319229590187360733544191994594869707714331873975449
52178856486730360250855218376629315433868751030662464046797
92937093469772528468011757947147166512152435404133191490416
62379371410408228902225087357647808337690948003983210745611
45687045287021997557751618765742763952553443287374734164531
45852893299326010312016761391559829086609861332167705597911
05215613850826759463538026285614903838507250897855705838724
49659903904311814242373052747205523182645833380301927933689
41333002995026564410490856532115413083850985103857380722832
07670767428187579815903755777474808788692079827442430868669
06201876284669292803039920577575111769507852806807547993017
22047807967084300522377435289623231007430240810484881231627
20936841694432593094616638077022074433667171760909894120864
29851843054317235434781840295992725379325881046570058485799
10932410525323549488828231695666943591839576583179961346342
08340828397695246816865625825963536531635766295748513828142
69603286954056733388899399413717268171807826546304347207339
35668869528756280020854585739047030891773673851859163735002
51957760959620164713390802215178934127168445639228483062049
30666580683109721538626401620997723858811498191312504063464
00767577318215869704292807511908855522623708661507760800651

39934662529734965835523308541540768555197945610801214995780
58418562053010315593167446406474338879090488536449022100972
71541065224624804293126599606409925169452156973395716618555
05135749297292670237728287161978930996259381758646342888408
53138933102054490270485556444249909288490760380933236016346
50142874212409602811687624242708597540960631386528199815652
00366175472358602009978307040404327446677073130088626731565
26035602186508147390749055766257971169402147723164779364656
23774315016467408466796245416985801276491432859164349833233
38565361720698275486225759313680820904323725581657553857160
71948712876678064438815319721125394157288775250404905382901
31435147127270914397227595612135035965523502394602395309928
22789883699715113831297562256458704438073465556034919794859
61992449455976243221481751411167149406708925978382307979100
56339886390867186697957387176618151963711927305352973712059
09578889478793546547847150611182723264663716358585968884463
39217539608168611631410176328155771748461786159502346351764
78547915862327401088544366654385988353407028200194454544617
73750579954972437089592018109735944234125124254397900090312
58351846844264687501405368598073628365903593454248027110391
39637950473220893400253606651628956090374354970460018382063
68680146254383101666087499787250539925637517678519243874344
37206872923323905750444178015698693283111304309213561600581
21842594220886557662181357368897040409566589710865888292361
64438525543472329573145822383696530724257735976232084519734
50606410160209709188367193916325390370546448243195840278844
79727032498249886561442752963805497655968589209771652324967
30638654080159385166303767164215545650853827341869974615270
49568193998895094071010793259209703588724471172123282271889
90543398458779845544281195301301588568445638914577378951818
88339478546565469189085580379090657294443617557387492553509
84073815264042654904766104724380470362406597942802032560269
84922608535332381874930646173980783677802048083531252098020
59579939538857419807117845468981872085801593354594013977583
09752314344113839928264275165966960772119278671384897927232
70797935779061764289333203584632650570545507293878464549700
84998059194684280341834694119441238185903679055221592592571
28697983034398362167392661845873494120286879935722337439619
72113106969534296502917127984081513247598447824546253439145
51668911747836191208650315997425701708532739253752241519925
67333437629564986362821857203338816194898532423260501712433
61930507304665115114836754111295691342976069443591326910757
89416683815549443935484507327290927869535849229588405261466
38632946401831942055149470707330973019817846396892365218911
81001906710419439895219252131812180753640540416516363059279
66248491924087640174074627814207698958809210026263381266659
73449629285508410110572796450231780309206940378888430305795
17647568661540624474105389905643442347561421462366640130069
03511076434903458275967683217803679790874318511990345694897
91887855640708065989941801552467204076296646993693236635065
36501117701055868582355811954302737246702288220571315639590

61822027640052204288908482254041966887584978480948804804531
68842877575361652520057158629595160885099803962695340286603
65420928832906521186092765576521399427875156924799554483049
60408342035664949958679770243634087828122915569941363579273
08113855650386504632881711947132973787269397260118254731758
05568507298463372949182573598560708416955751131747136868195
90716263857614668544618616367441291937469198270179365762316
27274242851618669771351464936198226636040729051607760103015
32925112772102255453238309477101692745197761359929665709233
51531295824879397991322837579720238982560189119063069027081
92127935896234473722810639917111975029763535242529850876912
33430206123870409178015883291874877723120751841704266668129
87442697693124661510035155106762505818472700736893694628439
4840868733993575255583583170
55359947722039227225423876928932200844350921818577101458295
13511381906756017382397217776898779213454899732222634652574
89935099549137644474027777583956137051269588783021487059246
17598489865075863183194186708153376785925896453521253497876
13552701597508011611524503843754837913945812148834125113809
99171425821993170789973409296543662081

I'm trying to move a part of SysTools to Win64. There is a class `TStDecimal` which is a fixed-point value with a total of 38 significant digits. The class itself uses a lot of ASM code.

```
function BigDecimal(aone: float; atwo: integer): string;
begin
 with TStDecimal.create do begin
  try
   //assignfromint(aone)
   assignfromfloat(aone) //2
   RaiseToPower(atwo)  //23
   result:= asstring
  finally
   free
  end;
 end;
end;
```

SysTools is hosted under Sourceforge:

http://www.sourceforge.net/projects/tpsystools

The class `TStDecimal` is defined in the unit `StDecMth`. It has the following description: `StDecMth` declares and implements `TStDecimal`. This is a fixed- point value with a total of 38 significant digits of which 16 are to the right of the decimal point.

1366556882568704.2292943165706246

☝ It is important to note that Infinity is not a real number, it is an idea. An idea of something without an end.

Infinity is not "getting larger", it is already fully formed. Sometimes students or people (including me) say it "goes on and on" which sounds like it is growing somehow. But infinity does not do anything, it just is.

```
Writeln('')
Writeln('Big Lotto Combination 1600 of 5000!')
Writeln('')
Writeln(BinominalCoefficient(5000, 1600));
Writeln('')
Writeln('Same Lotto Comb 6 of 45!')
Writeln(BinominalCoefficient(45, 6));     -->8145060
Writeln('Same Lotto Comb 39 of 45!')
Writeln(BinominalCoefficient(45, 39));  -->8145060
```

OK, 1/3 is a finite number (it is not infinite). But written as a decimal number the digit 3 repeats forever (we say "0.3 repeating"):

0.33333333333333333333... (etc.)

There's no reason why the 3s should ever stop: they repeat infinitely.
Okay, I hope you're not one of those people who denies 0.999... = 1, because it sounds like you're saying 0.333... doesn't exactly equal 1/3. If there are only a finite amount of 3s, I wouldn't argue a bit that they're not equal, but with an infinite amount, they are.

Test the script with **F9** / F2 or press Compile.

Conclusion And we can easily create much larger numbers than those! But none of these numbers are even close to infinity. Because they are finite, and infinity is ... not finite!

“Wise men speak because they have something to say; Fools, because they have to say something”. -Plato

Feedback @ max@kleiner.com

Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

http://www.softwareschule.ch/download/codesign_2015.pdf

http://www.softwareschule.ch/download/XXL_BigInt_Tutorial.pdf

http://www.mathsisfun.com/numbers/infinity.html

https://github.com/maxkleiner/maXbox3/releases

1.3 Appendix Study with BigInt Direct

```
// TODO: Copy a file in a connected share path
//this is 333^4096:


function GetBigIntDirect: string;
 //Unit mybigint
var mbResult: TMyBigInt;
    i: integer;
begin
  mbResult:= TMyBigInt.Create(333);
  try
   // Faktoren im Zaehler aufmultiplizieren  ---> 2^12=4096
   for i:= 1  to 12 do begin
     mbResult.Multiply(mbresult, mbresult);
     //writeln(inttostr(i)+': '+mbresult.tostring);
     end;
   Result:= mbResult.ToString;
  finally
   //FreeAndNil(mbResult);
   mbResult.Free;
  end;
end;


 TMyBigInt = class
   private
    Len: Integer;
    Value: AnsiString;
    procedure Trim;
    procedure Shift(k: Integer);
    procedure MultiplyAtom(Multiplier1: TMyBigInt; Multiplier2: Integer);
   public
    constructor Create(iValue: Integer = 0);
    procedure Add(Addend1, Addend2: TMyBigInt);
    procedure Multiply(Multiplier1, Multiplier2: TMyBigInt); overload;
    procedure Multiply(Multiplier1: TMyBigInt; Multiplier2: Integer); overload;
    function ToString: string;
    procedure CopyFrom(mbCopy: TMyBigInt);
   end;
```