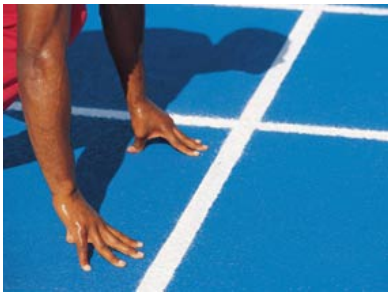
maXbox



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:

199987.740088484558199670720250704581668227404014653098834421683066491410778237138415987416773868407092963782727362267220...

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

again as you suppose the numbers go on.

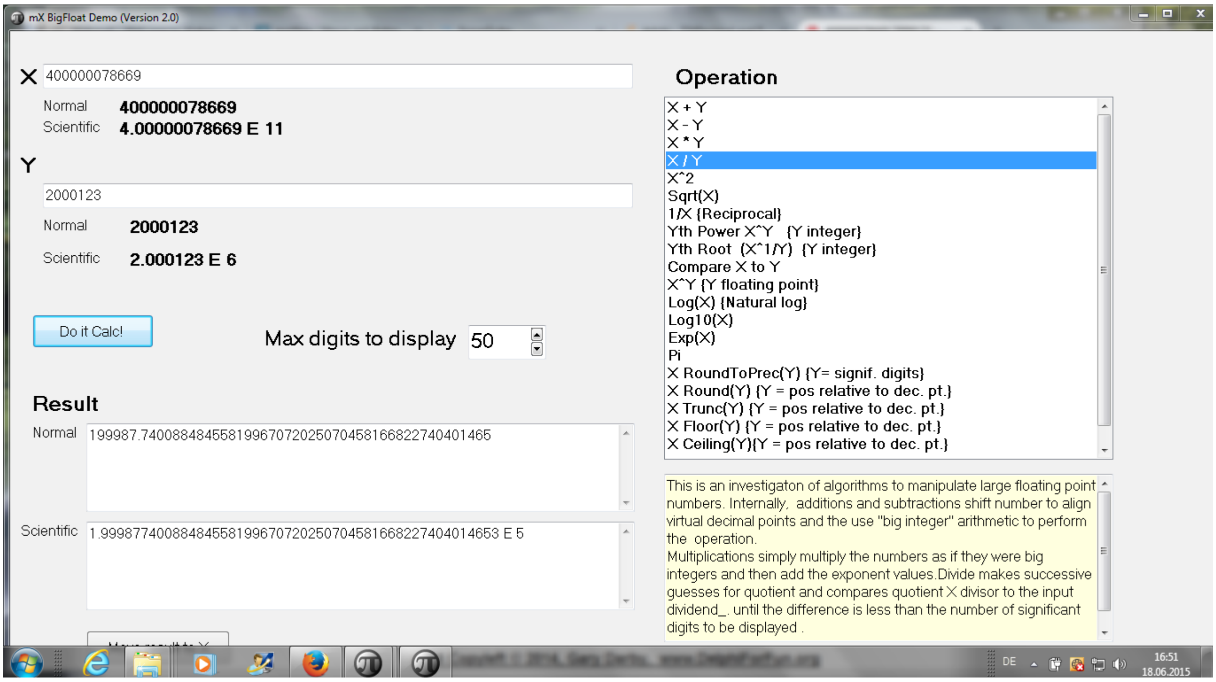
Use "Power Towers" to write them down. The decimal point is the most important part of a decimal number like above. Without it, we would be lost ... and not know what each position meant.
Dividing decimals is almost the same as dividing whole numbers, except you use the position of the decimal point 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 or more:

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:

199987.7400884845581996707202507045816682274040146530988344216830664914107782371384159874167738684070929637827273622672205

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 re-presented 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 (try also BigFact() or BigFibo())

1.1978571669969891796072783721689098736458938142546425857...
× 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):

11978571669969891796072783721987892755536628009582789845319
68000000000000000

All examples can be found online:

`..\examples\161_bigint_class_maxprove3.txt`

http://www.softwareschule.ch/examples/161_bigint_class_maxprove3.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)!

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
   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'n do anything, it just is.

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!

Feedback @ max@kleiner.com
Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

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

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

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