HighPower large integer calculator intended to investigate the properties of large numbers such as large exponentials and factorials. This application is written in Delphi 7 and can be easily ported to other languages. Currently there is no limit to the number of digits that software support. However exponential and factorials must be less than 1000000 . This free software a good desktop alternative software for **WolframAlpha** web based calculator. This software can calculate 2^{10000} in 1 seconds, 2^{100000} in 16 seconds . This software uses only the basic Delphi 32 bit integer. None of the extended floating point data type are used in this implementation.

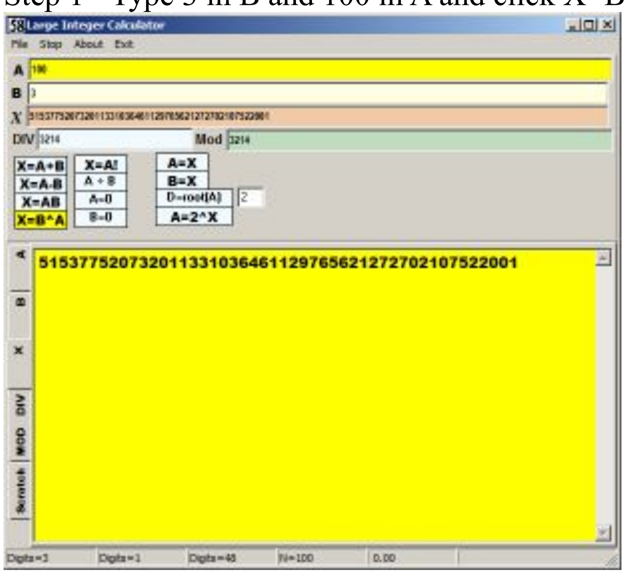
This application support the 4 basic arithmetic operations and log at base 2. At present this app does not support floating point numbers. Because it is provided as a basic tool that you can later add your own bells and whistles to it. Few examples provided

suppose we want to calculate these large values

1) $x=3^{100} + 2^{100}$
2) $x=3^{100} - 2^{100}$
3) $x=3^{100} * 2^{100}$
4) $x=3^{100} \div 2^{100}$
5) $x = \sqrt{2^{100} + 3^{100}}$
6) $x = \log_2^{(2^{100}+3^{100})}$
7) x=100!
8) calculate to 20 decimal accuracy $\sqrt{2}$

$x=3^{100} + 2^{100}$

Step 1 - Type 3 in B and 100 in A and click X=B^A



3^{100} You get X=515377520732011331036461129765621272702107522001

Step 2 . Copy this result into the windows Clipboard using standard WORD copy and paste

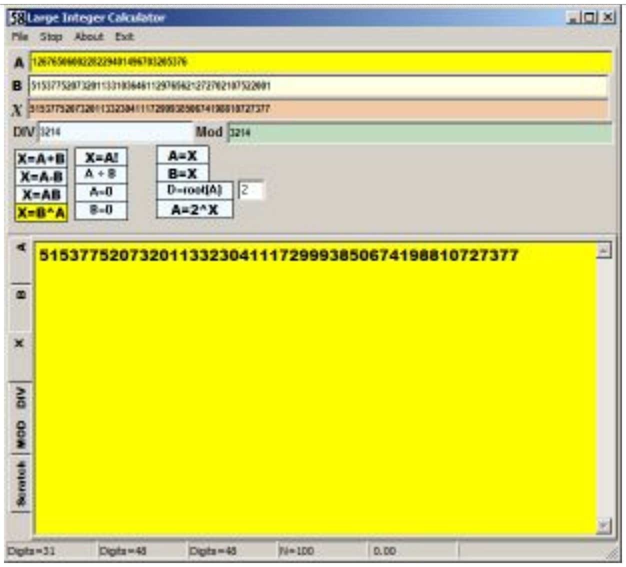
Step 3 . Type 2 in B and 100 in A and click X=B^A

You get X=1267650600228229401496703205376

Step 4 - Click on label A=X

Step 5 In B edit paste the value that was in clipboard

Step 6 - Click on label X=A+B . You should see final result



Therefore

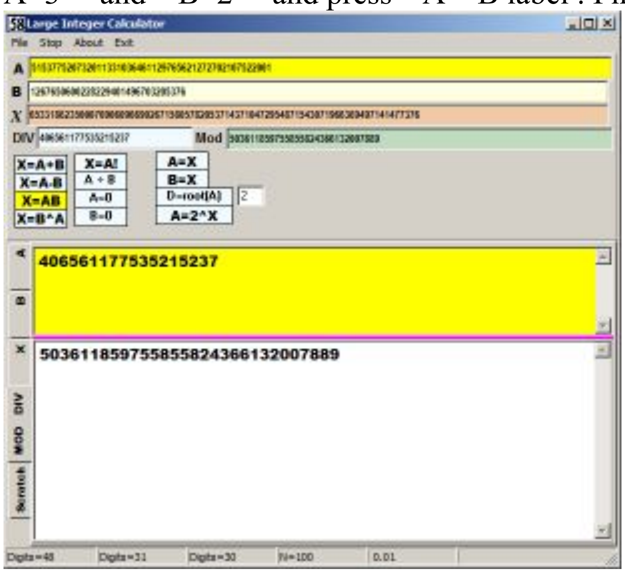
$3^{100} + 2^{100}$=515377520732011332304111729993850674198810727377

If you repeat the same procedure for subtraction, multiplication and division

$3^{100} - 2^{100}$=515377520732011329768810529537391871205404316625

6^{100}=$3^{100} * 2^{100}$=653318623500070906096690267158057820537143710472954871543071966369497141477376

x=$3^{100} \div 2^{100}$ to calculate the quotient and remainder as mentioned before make sure A=3^{100} and B=2^{100} and press A ÷ B label . Final result



Quotient=406561177535215237 Remainder=503611859755855824366132007889

Now suppose you want to have 10 decimal point accuracy. All you have to do is to repeat the above procedure but add 10 zeros to A such that

A=515377520732011331036461129765621272702107522001**0000000000**

Then the new quotient will be 4065611775352152373972797075 . You manually put the decimal point at

406561177535215237•3972797075

5) $x = \sqrt{2^{100} + 3^{100}}$

As in step 1 calculate X=515377520732011332304111729993850674198810727377
and press D=root(A) button
Therefore the square root of $2^{100} + 3^{100}$
will be calculated as 717897987691852589653139 and remainder will be 691521709936518478174056

6) $x = \log_2^{(2^{100}+3^{100})}$

As in step 1 calculate X=515377520732011332304111729993850674198810727377
and press A=2^X button x=158

7) x=100!

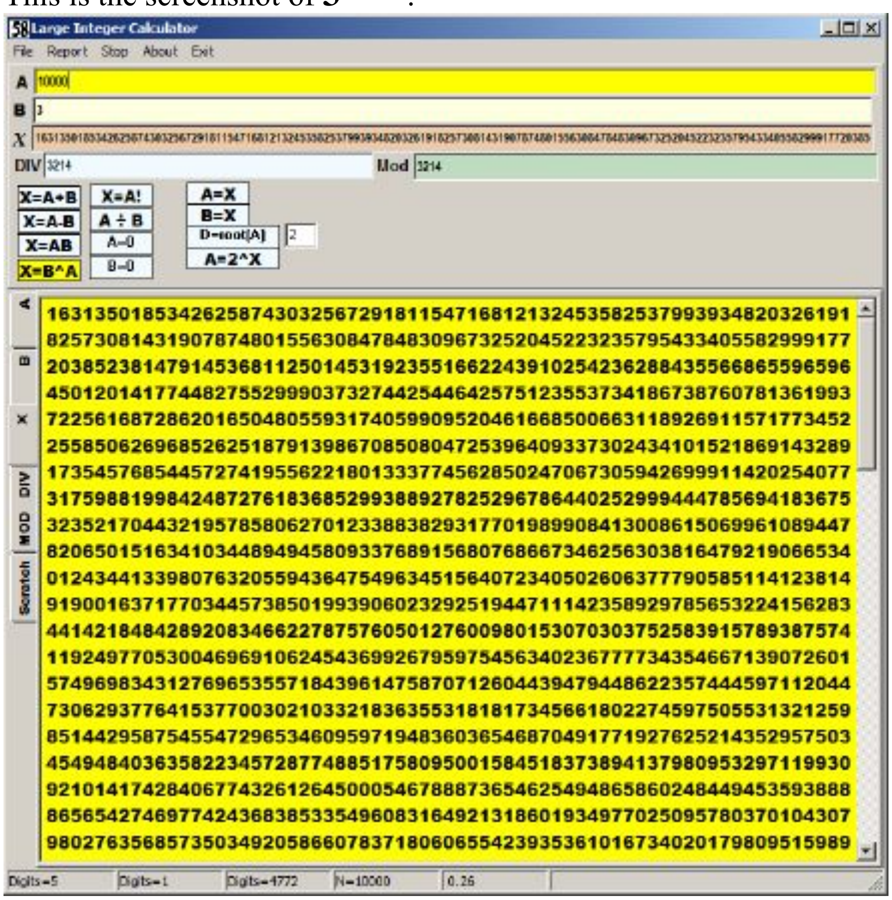
In A just type 100 and press the factorial label X=A!
100!=**93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000**

8) calculate to 20 decimal accuracy $\sqrt{2}$

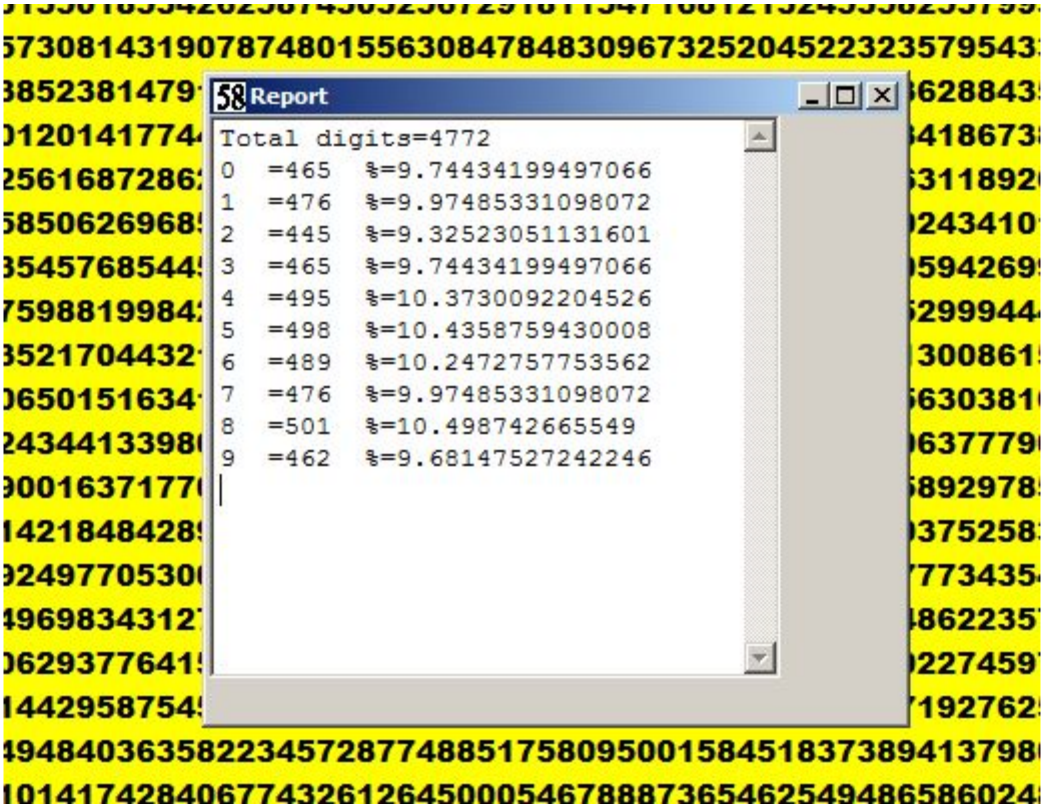
In A type 2 followed by 40 zeros .
20000000000000000000000000000000000000000
You will notice the DIV and MOD edit boxes will be modified as

141421356237309504880 . Therefore $\sqrt{2}$ =1•41421356237309504880

This is the screenshot of $\mathbf{3^{10000}}$.



You can get some basic report about the frequency of digits using the Report menu. For above the report will look like



Programming notes .

There are 13 math related functions and procedure are used to simulate very large numbers and their calculations.

```
function NormalizeStr(var s : string) : boolean;
procedure ReverseStr(q : string; var p : string);
procedure AddStrings(p,q : string; var R : string);
function CompareStrings(s,t : string) : integer;
function SubStrings(p,q : string; var r : string) : boolean;
procedure DivMod(Dividend: Integer; Divisor: Word; var Result, Remainder: Word);
function MulSingleDigit(i : Byte; p : string; var q : string) : boolean;
procedure ShiftString(var s : string; n : integer);
procedure MulStrings(p,q : string; var R : string);
function Power2(n : integer): string;
function DivStrings(var a,b : string; var r,s : string) : boolean;
Procedure Root(S : string; n : integer; var p,q : string);
procedure Factorial(n : integer; var q : string);
procedure LogStringBase2(S : string ; var n : integer);
```

function NormalizeStr(var s : string) : boolean;
The ascii strings are normalized to reflect the actual value of each digit. The ascii order of '7' which is 55 is modified to be actual binary 7. All the ascii values subtracted 48 for normalization. If any non numerical character detected the result of normalization will be set to False to prevent calculations to proceed.

procedure ReverseStr(q : string; var p : string);
For example when you enter
78001289 the LSB is 9 but in Delphi strings 7 is the first data . This procedure reverse it for further processing as 98210087

procedure AddStrings(p,q : string; var R : string)
After 2 strings p and q are normalized , the string R will contain their sum. To display this string correctly it must be reversed again.

function CompareStrings(s,t : string) : integer;
is used to see which string is greater than other . If result=-1 then s<t , if result=0 then s=t otherwise s>t

function SubStrings(p,q : string; var R : string) : boolean;
if p>q then string R will contain the difference between p-q

function MulSingleDigit(i : Byte; p : string; var q : string) : boolean;
As in normal multiplication digits of multiplicand must be multiplied by digits of multiplier

procedure DivMod(Dividend: Integer; Divisor: Word; var Result, Remainder: Word);
This ASM procedure allows quick calculation of DIV and MOD in 1 procedure. Used in **MulSingleDigit**

procedure ShiftString(var s : string; n : integer);
procedure MulStrings(p,q : string; var R : string);
Given 2 previously normalized strings p and q their multiplication result is stored in string R . To multiply 2 numerical strings it is necessary the resultant strings to be shifted to left to have the multiply by 10 effect. Mulstrings and Shiftstring work closely together to achieve 2 numerical strings regardless of their lengths to be multiplied at any desired accuracy.

function Power2(n : integer): string;
Division and Square root unlike other arithmetic operations are based of initial trial and guess.This software uses powers of tables for calculating division and square roots. To make future divisions faster the powers of 2 stored in stringlist so next time they can be looked up instead of calculated.

function DivStrings(var a,b : string; var r,s : string) : boolean;
2 previously normalized strings a,b such that a>b are divided . r will contain the quotient and s contain the remainder . To better understand the division algorithm that is used for division lets take a look at this example.
Lets calculate quotient and remainder of $3759 \div 53$.
1) We start multiplying 53 by powers of 2 such that result will be greater than 3759
53*2=106   53*4=212   53*8=424   53*16=1024   53*32=1696   53*64=3392   53*128=6784

Therefore the initial multiplicand that we will use is(2^8) F=64 . Now we try to add lower powers of 2 to F and multiply the result by 53 such that the result will be less or equal to 3759.

So we try (32) 2^5 first . F=64+32=96   96*53=5088 . but 5088>3759 therefore we need to reduce power
now try 16 , F=64+16=80   53*80=4240 but 4240>3759   therefore we need to reduce power
now try 8   F=64+8=72   53*72=3816 but 3816>3759 therefore we need to reduce power
now try 4   F=64+4=68   53*68=3604. Since 3604<3759 Therefore we update F=64+4=68
now try 2   F=68+2=70   53*70=3710 . since 3710<3759 again we update F=68+2=70
now try 1   F=70+1=71   53*71=3763   but 3763>3753 , we can not add to F .

As you see final quotient is therefore F=2^8+2^2=70

Procedure Root(S : string; n : integer; var p,q : string);
Square root of string S is calculated such that p is root and q remainder . var n for time being is 2 . Later other higher roots can be calculated. Square root similar to DivStrings is based on initial trial and guess.
To better understand the square root algorithm lets take a look at this example $\sqrt{933}$

1)We start calculating powers of 2 such that the result will be greater or equal to 933

$2^1=1$, $2^2=4$, $2^3=8$, $2^4=16$, $2^6=64$, $2^8=256$, $2^9=512$, $2^{10}=1024$
since 1024 is the first number greater than 933 we selected the square root of previous number F=16 . Like division algorithm we add powers of 2 to F and square the result. If result is bigger then we go to next power of 2 until we reach 1 . Therefore we try F=16+8=24 then 24*24=576. Since 576<933 we use the new value F=24 and add next power of 2 which is 4. Therefore F=24+4=28 and 28*28=784 . Since 784<933 we continue adding . F=28+2=30. 30*30=900. Finally we try $2^0=1$. We have F=30+1=31 . But 31*31=961 and 961>933 . therefore we keep the previous value as final result F=30 as $\sqrt{933}=30$

Some useful powers of 2 and 3 exponents

N	Exp	Value
2	**40**	1099511627776
2	**50**	1125899906842624
2	**60**	1152921504606846976
2	**100**	1267650600228229401496703205376
2	**200**	1606938044258990275541962092341162602522202993782792835301376
2	**500**	3273390607896141870013189696827599152216642046043064789483291368096133796404674554883270092325904157 150886684127560071009217256545885393053328527589376
2	**1000**	1071508607186267320948425049060001810561404811705533607443750388370351051124936122493198378815695858 1275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954 1821530464749835819412673987675591655439460770629145711964776865421676604298316526243868372056680693 76
3	**40**	12157665459056928801
3	**50**	717897987691852588770249
3	**60**	42391158275216203514294433201
3	**100**	515377520732011331036461129765621272702107522001
3	**200**	265613988875874769338781322035779626829233452653394495974574961739092490901302182994384699044001
3	**500**	3636029179586993684238526707954331911802338502600162304034603583258060019158389548419850826297938878 3308179702534403855752855931517013066142992430916562025780021771247847643450125342836565813209972590 371590152578728008385990139795377610001
3	**750**	6933316722256889501243270035054617317131441484599852861651231094188457364439209085813870473832966342 4402738132990122325290494654915439166993068069352082479188773679332625787214038973363408389313034602 3098769760762756624392752628692941852839077472128238547401603196418915214887575559449969127678784601 9565102537919126221646506678418775883216359004272415616249
3	**1000**	1322070819480806636890455259752144365965422032752148167664920368226828597346704899540778313850608061 9639097776968725823559509545821006189118653427252579536740276202251983208038780147742289648412743904 0011758861804112894781562309443806156617305408667449050617812548034440554705439703889581746536825491 6136220830268563778582290228416398307887896918556404084898937609373242171846359938695516765018940588 109060426089671438864102814350385648747165832010614366132173102768902855220001

Some useful Factorial table

20!	2432902008176640000
50!	30414093201713378043612608166064768844377641568960512000000000000
100!	9332621544394415268169923885626670049071596826438162146859296389521759999322991560894146397615651828625369 79208272237582511852109168640000000000000000000000000
200!	7886578673647905035523632139321850622951359776871732632947425332443594499634033429203042840119846239041772 1213891963883025764279024263710506192662495282993111346285727076331723739698894392244562145166424025403329 1864131227428294853277524242407573903240321257405579568660226031904170324062351700858796178922222789623 7038973747200000000000000000000000000000000000000000
200! DIV 3^{750} =113749003393006914 http://www.wolframalpha.com/input/?i=200!+div+(3%5E750)	

What is next : Factorization of large integer and search for large prime numbers

Other large integer sources on web
https://torry.net/quicksearchd.php?String=integer&Title=Yes

http://rvelthuis.de/programs/bigintegers.html

http://delphiforfun.org/programs/library/big_integers.htm