The program PiW can compute Pi to a million or so decimal places. It is written in Borland C++ for Windows, so it requires an IBM PC compatible computer running Microsoft Windows 3.0 or later. It is recommended that Windows be run in the 386 enhanced mode. With 16 megabytes of RAM it can compute Pi to 524,200 decimal places, with 32 megabytes of RAM 1,048,500 places. Run the executable file PiW.Exe and you will get the message:

```
PiW - Compute Pi to a million or so decimal places in Windows
Version 1.31, last revised: 1994-07-29, 0600 hours
Copyright (c) 1981-1994 by author: Harry J. Smith,
19628 Via Monte Dr., Saratoga, CA 95070.  All rights reserved.

Will write file: PiW.Out containing Pi.

Will write file: PiW.Log containing a log of operations.

Will write file: PiReset.Txt containing a restart file.

Will write file: PiReset.Old containing a restart file backup.

How many decimal places do you need? (n < 0 to restart) (0 to quit): _
```

Type in a number like 2000 and press Enter. The program will respond with:

```
Use algorithm b? (Y/N): _
```

Type a single letter (y, Y, n, or N) and press Enter. The program will respond with:

```
How many digits between commas in output? (0 for no , < 0 for digits only): _
```

Three would be the most common input here, but zero and five are also common. A negative input causes nothing but the digits of Pi to be output. An input of -32768 is treated as minus zero.

Type a number and press Enter. The program will start computing Pi and output a log of its operations with diagnostic and timing data to the screen and to the file PiW.Log. If you answer

with a character other than y or Y above the program will use algorithm a instead of algorithm b. For a request of 2000 decimal places, algorithm b, and no commas, the output looks like:

```
 Start of PiW.Cpp 1994-07-29 06:30:21.49
 Trying to compute Pi to about 2023 decimal places.
 T = 0  DT = 0 sec.  Starting Pi
   Starting to compute Pi by algorithm b
 T = 2.31  DT = 2.31 sec.  Starting Iteration
   Start of iteration number 1 of 5
 T = 19.17  DT = 16.86 sec.  Starting Iteration
   Start of iteration number 2 of 5
 T = 34.94  DT = 15.77 sec.  Starting Iteration
   Start of iteration number 3 of 5
 T = 52.68  DT = 17.74 sec.  Starting Iteration
   Start of iteration number 4 of 5
 T = 70.64  DT = 17.96 sec.  Starting Iteration
   Start of iteration number 5 of 5
 T = 89.26  DT = 18.62 sec.  All done but...
   All done but inverting alpha
 T = 91.67  DT = 2.41 sec.  Pi is done
   All done computing Pi

 T = 91.73  DT = 0.06 sec.  Writing Pi to disk

 Pi = m.n E+0, m.n =

3.14159265358979323846264338327950288419716939937510582097494459230781640628620
9986280348253421170679821480865132823066470938446095505822317253594081284811174
5028410270193852110555964462294895493038196442881097566593344612847564823378678
3165271201909145648566923460348610454326648213393607260249141273724587006606315588
1748815209209628292540917153643678925903600113305305488204665213841469519415116
0943305727036575959195309218611738193261179310511854807446237996274956735188575
2724891227938183011949129833673362440656643086021394946395224737190702179860943
7027705392171762931767523846748184676694051320005681271452635608277857713427577
8960917363717872146844090122495343014654958537105079227968925892 3E+0 (700) [2051]

 T = 93.82  DT = 2.09 sec.  All done
 End of PiW.Cpp 1994-07-29 06:31:55.41
```

It is recommended that, if you compute Pi to more places than you can verify to be correct by an independent source, you should compute Pi to this number of places using both algorithm a and algorithm b and use the DOS program FC (File Compare) to verify that the same value was computed by both algorithms.

These algorithms are documented in Scientific American, February 1988, Ramanujan and Pi, by Jonathan M. Borwein and Peter B. Borwein.

Algorithm a:

Let y[0] = SqRt(1/2), x[0] = 1/2

y[n] = (1 - SqRt(1 - y[n-1]^2)) / (1 + SqRt(1 - y[n-1]^2))

x[n] = ((1 + y[n])^2 * x[n-1]) - 2^n * y[n]

Algorithm b:

Let y[0] = SqRt(2) - 1, x[0] = 6 - 4 * SqRt(2)

y[n] = (1 - SqRt(SqRt(1 - y[n-1]^4))) / (1 + SqRt(SqRt(1 - y[n-1]^4)))

x[n] = ((1 + y[n])^4 * x[n]) - 2^(2n+1) * y[n] * (1 + y[n] + y[n]^2)

For both algorithms, x_n converges to 1/Pi. Algorithm a is quadratically convergent and algorithm b is quartically convergent. The following table shows how many iterations are needed to compute Pi to a given number of significant digits:

Iterations of algo. a	Iterations of algo. b	Digits matching Pi
1	0	
2	3	($1/x_2$ = 3.140, algo. a)

3	1	8
4		19
5	2	41
6		84
7	3	171
8		345
9	4	694
10		1392
11	5	2788

Algorithm b converges to a given number of digits about twice as fast as algorithm a, but takes about twice as much work per iteration. Using PiW, algorithm b is 10 to 20 percent faster than algorithm a.

I used PiW to compute Pi to 500,000 decimal digits. This was done on an IBM AT compatible 33 MHz 486 DX computer using Windows 3.1 with 16 megabytes of RAM and 22 mega bytes of virtual memory. It took 37.3 hours for algorithm a and 31.0 hours for algorithm b. The results were the same. The divide and square root routines have been improved since these runs, so the program is a little faster now (10 to 20%).

The distribution disk for PiW includes the executable file PiW.Exe, all the source code, this document, and the file PiW500K.Out containing Pi computed to 500,000 decimal digits. Read the READ-ME and the WHATFOR files for a complete description of what's on the disk.

The first time I tried to compute Pi to more places than is normally needed I computed it to 50 decimal places using paper, pencil, and a hand calculator. The equation used was

Pi = 16 * ArcTan(1/5) - 4 * ArcTan(1/239)

ArcTan(1/x) = 1/x - 1/(3 * x^3) + 1/(5 * x^5) - ...

This was done in the early 1970's. The equation was derived in 1706 by John Machin (1685-1751). After I got my Apple II+ computer in 1979, I computed Pi to 15,300 decimal places using Apple Pascal and the equation

Pi = 48 * ArcTan(1/18) + 32 * ArcTan(1/57) - 20 * ArcTan(1/239)

This is the same equation I used in 1989 to compute a 114,632 decimal place value using Borland Turbo Pascal on an IBM AT compatible computer. The equation is due to Carl Friedrich Gauss (1777-1855).

In 1989, Pi was computed to 1,011,196,691 decimal digits. This was done by the Chudnovsky brothers (David and Gregory), Columbia University, using super computers.

There has been several major brakethroughs that has allowed the calculation of Pi to get into the million digit range on a personal computer. A major limitation in Pascal is that array size is limited to 64K bytes and arrays use 16-bit integer as their index. In Borland C and C++, arrays are only limited by the amount of memory in your system and arrays can use long integer (0 - 2,147,483,647) as their index. This Pascal limitation forced me to convert my Multiple-precision packages from Pascal to C++.

The equations used to compute Pi were linearly convergent. This means if we want twice the accuracy, we need to compute twice as many terms at twice the precision. This was solved by using algorithm a or b.

The multiply, divide, and square root routines that would be used in algorithm a and b were running in time of the order of n squared $O(n^2)$. This meant that if it took 20 seconds to compute pi to 1000 digits it might take 230 days to compute it to 1,000,000 places. One way to change the order of the running time of the multiply is to use a fast Fourier transform (FFT) to perform the convolution of the digits between the two numbers and to perform the carry out of the digits after the convolution. This makes the running time of the multiply approximately O(n Log n). The divide routine can be changed to an iterative method that uses the multiply. The square root routine can be changed to an iterative method that uses the divide.

The last break through was getting an operating system that would allow you to easily use large arrays in extended memory. This was accomplished by using MS Windows and Borland C++ for

Windows.

We now have a package on a PC that computes Pi in the 1,000,000 decimal digit range and runs in a time of approximately $O(n \text{ Log}^3 n)$.