Computing π

Kiyoshi Akima
http://kiyoshiakima.tripod.com/funprogs

2006.04.27

Contents

1 Computing π

The Viennese mathematician L. K. Schulz von Strassnitzky (1803–1852) used an arctangent formula to program the forerunner of the computer, a calculating prodigy. This one was Johann Martin Zacharias Dase (1824–1861), about whom we shall have more to say below. In 1844 he calculated π correct to 200 places in less than two months.

But just how does one go about computing the value of π?

1.1 The Ancients

By about 2000 B.C.E., the Babylonians had arrived at the value

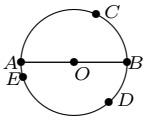
$$\pi = 3\ 1/8 \tag{1}$$

and the Egyptians at the value

$$\pi = 4(8/9)^2 \tag{2}$$

How did these ancient people arrive at these values? Nobody knows for certain, but it's not hard to make a good guess.

Obviously, the easiest way is to take a circle, to measure its circumference and diameter, and to find π as the ratio of the two. Let us try to do just that, imagining that we are in Egypt in 3,000 B.C.E. There is no National Institute of Standards and Technology; no Royal Academy; no calibrated measuring tapes. We are not allowed to use the decimal system or numerical division of any kind. No compasses, no pencil, no paper; all we have is stakes, rope, and sand.



So we find a fairly flat patch of wet sand along the Nile, drive in a stake, attach a piece of rope to it by loop and knot, tie the other end to another stake with a sharp point, and keeping the rope taut, we draw a circle in the sand. We pull out the central stake, leaving a hole O. Now we take a longer piece of rope, choose any point A on the circle and stretch the rope across the hole O until it intersects the circle at B. We mark the length AB on the rope (with charcoal); this is the diameter of the circle and our unit of length. Now we take the rope and lay it into the circular groove in the sand, starting at A. The charcoal mark is at C; we have laid off the diameter along the circumference once. Then we lay it off a second time from C to D, and a third time from D to A, so that the diameter goes into the cirumference three (plus a little bit) times.

If, to start with, we neglect the little bit, we have, to the nearest integer,

$$\pi = 3 \tag{3}$$

To improve our approximation, we next measure the little left-over bit EA as a fraction of our unit distance AB. We measure the curved length EA and mark it on a piece of rope. Then we straighten the rope and lay it off along AB as many times as it will go. It will go into our unit distance AB between 7

and 8 times. (Actually, if we cheat, we find that 7 is much nearer the right value than 8. However, that would be difficult to ascertain by our measurement using thick, elastic ropes with coarse charcoal marks for the roughly circular curve in the sand whose surface was judged "flat" by arbitrary opinion.)

We have thus measured the length of the arc EA to be between 1/7 and 1/8 of the unit distance AB; and our second approximation is therefore

$$3\ 1/8 < \pi < 3\ 1/7 \tag{4}$$

for this, to the nearest simple fractions, is how often the unit rope length AB goes into the circumference $ABCD$.

And indeed, the values

$$\pi = 3, \quad \pi = 3\ 1/7, \quad \pi = 3\ 1/8$$

are the values most often met in antiquity.

For example, in the Old Testament, we find the following verse:

> *Also, he made a molten sea of ten cubits from brim to brim, round in compass, and five cubits the height thereof; and a line of thirty cubits did compass it round about.*

The molten sea,we are told, is round; it measures 30 cubits round about (in circumference) and 10 cubits from brim to brim (in diameter); thus the biblical value of π is $30/10 = 3$.

Returning to the determination of π by direct measurement using primitive equipment, it can probably safely be said that it led to values no better than (4).

From now on, man had to rely on his wits rather than on ropes and stakes in the sand. And it was by his wits, rather than by experimental measurement, that he found the circle's perimeter.

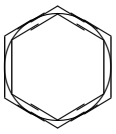
1.2 Archimedes of Syracuse

When Newton said "If I have seen further than others, it is because I stood on the shoulder of giants," one of the giants he must have had in mind was Archimedes of Syracuse, the most brilliant mathematician, physicist, and engineer of antiquity.

Little is known about his life. He was born in Syracuse and apparently spent most of his life there. He studied at the University of Alexandria either under Euclid's immediate successors, or perhaps under Euclid himself. He was a kinsman and friend of Hieron II, king of Syracuse, for whom he designed the machines of war used against the Roman aggressors, and whose crown was involved in the discovery of the law of upthrust that bears his name. Hieron suspected (correctly) that his crown was not pure gold and asked Archimedes to investigate without damaging the crown. Archimedes is said to have pondered the problem while taking a bath, and to have found the answer as he observed the water level rising on submerging his body into the bath. Shouting *Heureka!*

(I have found it), says this legend, he ran naked through the streets of Syracuse to tell Hieron of his discovery.

His book *On Floating Bodies* goes far beyond Archimedes's Law, and includes complicated problems of buoyancy and stability. Likewise, *On the Equilibrium of Planes* goes beyond the principle of the lever and solves complicated problems such as finding the center of gravity of a parabolic segment. In these, as in his other works, Archimedes used the Euclidean approach: From a set of simple postulates, he deduced his propositions with unimpeachable logic. As the first writer who consistently allied mathematics and physics, Archimedes became the father of physics as a science. (Aristotle's *Physics* was published a century earlier, but is only a long string of unfounded speculations, totally devoid of any quantitative relations.)



He was also the first to give a method of calculating π to any desired degree of accuracy. It is based on the fact that the perimeter of a regular polygon of n sides inscribed in a circle is smaller than the circumference of the circle, whereas the perimeter of a similar polygon circumscribed about the circle is greater than its circumference. By making n sufficiently large, the two perimeters will approach the circumference arbitrarily close, one from above, the other from below. Archimedes started with a hexagon, and progressively doubling the number of sides, he arrived at a polygon of 96 sides, which yielded

$$3\ 10/71 < \pi < 3\ 1/7 \tag{5}$$

or in decimal notation

$$3.14084 < \pi < 3.142858 \tag{6}$$

That Archimedes did this without trigonometry, and without decimal (or other positional) notation is an illustration of his tenacity.

Though later investigators found closer numerical approximation, the polygonal method remained unsurpassed for 19 centuries before the discovery of the differential calculus led to a totally new approach to the problem.

1.3 The Digit Hunters

James Gregory (1638–1675), a leading contributor to the discovery of the differential and integral calculus, was a mathematician occasionally dabbling in astronomy. He studied mathematics at Aberdeen and later in Italy. He worked on problems far ahead of his time; for example, in Italy he wrote the *Vera circuli et hyperbolae quadratura* (True quadrature of the circle and the hyperbola), which included the basic idea of the distinction between algebraic and transcendental functions, and he even attempted to prove the transcendence of π, a task that was not crowned with success until 1882. He was familiar with the series expansion of $\tan x$, $\sec x$, $\arctan x$, and the logarithmic series. In 1668 he returned to Scotland, where he became Professor of Mathematics at St. Andrews University, and in 1674 he was appointed to the first Chair of Mathematics at

the University of Edinburgh, but he died suddenly the next year at the age of only 36.

Among the many things that Gregory discovered, the most important for the computation of π is the series for the arctangent which still bears his name. He found that the area under the curve $1/(1+x^2)$ in the interval $(0, x)$ was $\arctan x$; in modern symbols,

$$\int_0^x \frac{dx}{1+x^2} = \arctan x \tag{7}$$

By the simple process of long division in the integrand, he found the Gregory series

$$\arctan x = \frac{x}{1} - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \cdots \tag{8}$$

From here it was a simple step to substitute $x = 1$; since $\arctan(1) = \pi/4$, this yields

$$\pi = 4(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots) \tag{9}$$

which was the first infinite series ever found for π.

Gregory discovered the series (8) in 1671, reporting the discovery in a letter of February 15, 1671, without derivation. Leibniz found the series (8) and its special case (9) in 1674 and published it in 1682, and the second series is sometimes called the Leibniz series. Gregory did not mention the special case (9) in his works. Yet is is unthinkable that the discoverer of the series (8), a man who had worked on the transcendence of π, should have overlooked the obvious case of substituting $x = 1$ in his series. More likely he did not consider it important because its convergence (a concept also introduced by Gregory) was too slow to be of practical use for numerical calculations. If this was the reason why he did not specifically mention it, he was of course right.

In 1706, John Machin, Professor of Astronomy in London, used the following statagem to make the Gregory series rapidly convergent and convenient for numerical calculations as well.

For $\tan\beta = 1/5$, we have

$$\tan 2\beta = \frac{2\tan\beta}{1-\tan^2\beta} = \frac{5}{12} \tag{10}$$

and

$$\tan 4\beta = \frac{2\tan 2\beta}{1-\tan^2 2\beta} = \frac{120}{119} \tag{11}$$

This differs only by $1/119$ from 1, whose arctangent is $\pi/4$; in terms of angles, this difference is

$$\tan(4\beta - \pi/4) = \frac{\tan 4\beta - 1}{1+\tan 4\beta} = -\frac{1}{239} \tag{12}$$

and hence

$$\arctan(1/239) = 4\beta - \pi/4 = 4\arctan(1/5) - \pi/4 \tag{13}$$

Substituting the Gregory series for the two arctangents, Machin obtained

$$\frac{\pi}{4} = 4\Big(\frac{1}{5} - \frac{1}{3\cdot 5^3} + \frac{1}{5\cdot 5^5} - \cdots\Big) - \Big(\frac{1}{239} - \frac{1}{3\cdot 239^3} + \frac{1}{5\cdot 239^5} - \cdots\Big) \quad (14)$$

This was a neat little trick, for the second series converges very rapidly, and the first is well suited for decimal calculations because successive terms diminish by a factor involving $1/5^2 = 0.04$. From this, Machin calculated π to 100 decimal places in 1706.

To return to Johann Martin Dase. He was born in 1824 in Hamburg, had a fair education, and was afforded every opportunity to develop his powers, but made little progress; all who knew him agree that except for calculating and numbers, he was quite dull. He always remained completely ignorant of geometry, and never learned any other language than German. His extraordinary calculating powers were timed by renowned mathematicians: He multiplied two 8-digit numbers in his head in 54 seconds; two 20-digit numbers in 6 minutes; two 40-digit numbers in 40 minutes; and two 100-digit numbers (also in his head!) in 8 hours and 45 minutes.

To achieve feats like these, he must have had a photographic memory. He could give the number of sheep in a flock, books in a case, etc., after a single glance; after a second's look at some dominos, he gave the sum of their points correctly as 117; and when shown a randomly selected line of print, he gave the number of letters correctly as 63. One may speculate that perhaps he achieved these feats by taking a glance at a flock of sheep, for instance, and then rapidly counting them from the photographic image in his mind. He could, for example, in half a second memorize a twelve-digit number and then instantly name the digit occupying a particular position, so that whatever the mechanism that enabled his brain to do this, it must have been close to photography.

Dase gave exhibitions of his extraordinary calculating powers in Germany, Austria, and England, and it was during an exhibition in Vienna in 1840 that he made the acquaintance with Schulz von Strassnitzky, who urged him to make use of his powers for the calculation of mathematical tables. He was then 16 years old, and gladly agreed to do so, thus becoming acquainted with many famous mathematicians of his age, including Carl Friedrich Gauss. When he was 20, Strassnitzky taught him the use of the formula

$$\pi/4 = \arctan(1/2) + \arctan(1/5) + \arctan(1/8) \quad (15)$$

with a series expansion for each arctangent, and this is what he used to calculate π to 205 decimal places (of which all but the last five turned out to be correct). This stupendous feat he finished in just under two months.

Then he came back for more. He calculated the natural logarithms of the first 1,005,000 numbers, each to 7 decimal places, which he did in his spare time in 1844-1847 when employed by the Prussian Survey. In the next two years he compiled a table of hyperbolic functions, again in his spare time. He also offered to make tables of the factors of all numbers from 7,000,000 to 10,000,000; and

on the recommendation of Gauss, the Hamburg Academy of Sciences agreed to assist him financially so that he could devote himself to this work, but he died in 1861, after he had finished about half of it.

It would thus appear that Carl Friedrich Gauss, who holds so many firsts in all branches of mathematics, was also the first to introduce payment for computer time.

1.4 The Computer Age

The first computer calculation of π was apparently made in September 1949 on ENIAC (Electronic Numerical Integrator And Computer) at the Ballistic Research Labs; it calculated π to 2,037 places in 70 hours, a pitifully long time by today's standards (the program described in this document can do it in about three seconds on the author's laptop computer). Like many other computer evaluations, this one was programmed in accordance with Machin's formula (14).

In November 1954 and January 1955, NORC (Naval Ordnance Research Calculator) at Dalhgren, Virginia, was programmed to compute π to 3,089 significant places; the run took only 13 minutes.

This record was broken at the Ferranti Computer Centre, London, in March of 1957, when a Pegasus computer computed 10,021 decimal places in 33 hours. The program was based on an arctangent formula similar to, but not identical with, the one used by Strassnitzsky (15). However a subsequent check revealed that a machine error had occurred, so that "only" 7,480 decimal places were correct. The run was therefore repeated in March 1958, but the correction was not published.

Then, in July 1958, an IBM 704 at the Paris Data Processing Center was programmed according to a combination of Machin's formula (14) and the Gregory series (8); it yielded 10,000 places in 1 hour and 40 minutes.

A year later, in July 1959, the same program was used on an IBM 704 at the Commisariat à l'Energie Atomique in Paris, and 16,167 places were obtained in 4.3 hours.

Machin's formula (14) was also the basis of a program run on an IBM 7090 at the London Data Centre in July 1961, which resulted in 20,000 decimal places and required only 39 minutes of processing time.

By this time the limit of then available computer memories had almost been reached. Further substantial increases in the number of decimal places could have been obtained only by modifying the programs to use more machine time and therefore to run into unreasonable costs.

But in July 1961, Shanks and Wrench increased the speed of the computation by a factor of about 20. In part, this was due to a faster computer (an IBM 7090 at the IBM Data Processing Center, New York), but they also used several tricks in programming it; in particular, they abandoned Machin's formula in favor of the formula

$$\pi = 24 \arctan(1/8) + 8 \arctan(1/57) + 4 \arctan(1/239) \tag{16}$$

which was found by Störmer in 1896. The run resulted in 100,265 decimal places, of which the first 100,000 were published by photographically reproducing the printout with 5,000 decimals per page. The time required for computing the first term in (16) was 2 hours and 7 minutes, for the second term 3 hours and 7 minutes, and for the third term 2 hours and 20 minutes. To this must be added 42 minutes for converting the final result from binary to decimal digits, so that the total time required was 8 hours and 43 minutes.

Subsequently, π was computed to 250,000 decimal places on an IBM 7030 at the Commisariat à l'Energie Atomique in Paris in February 1966, and a year later, in February 1967, a CDC 6600 was programmed by J. Gilloud and J. Filliatre, at the same institution, to yield 500,000 decimal places. The program was again based on Störmer's formula (16); the running time was 28 hours and 10 minutes (of which 1 hour and 35 minutes were used for conversion). These quarter- and half-million digit values were published in reports of the Commissariat à l'Energie Atomique in Paris.

And yet the digits continued to come. The one-million decimal digit mark was reached in 1973. The two-million mark was reached in 1981. Then the pace accelerated; 1982 saw first four and then eight million decimal digits. In 1986, a Cray-2 raised the bar to 29,360,111 digits.

Using a faster scheme, which is beyond the scope of this document, Yasumasa Kanada at the Laboratory for Computational Tools at the University of Tokyo computed 1,073,740,799 decimal digits in November 1989. He subsequently computed 3,221,220,000 digits in June 1996 and 4,294,960,000 digits two months later.

Then, in November 2002, Kanada again beat his own record, computing a whopping 1,241,100,000, 000 decimal digits. The run took about 400 hours on a Hitachi SR8000/MPP.

2 Rolling Our Own

Now it's time to write our own program to compute π. We're not going for the world record here; our program won't be the fastest nor the least memory intensive possible. Instead, as an example of how π *could* be computed, we'll strive for clarity.

We're also going to do this in BCPL.

The choice of a language without floating-point capabilities might seem strange at first. But a moment of thought reveals that floating-point arithmetic is of not much value for anything beyond about a dozen decimal digits. For anything beyond that, we're going to have to do the arithmetic ourselves, and that's no harder in BCPL than it is in any other language.

2.1 The Program

The program is quite straightforward. Other than some housekeeping, we first compute π and then print it.

8a ⟨* 8a⟩≡

```
GET "libhdr"

LET start() = VALOF {
    ⟨local procedures 10d⟩

    ⟨local variables 8b⟩

    ⟨determine number of digits 8c⟩
    ⟨allocate memory 10b⟩
    ⟨compute π 12d⟩
    ⟨print π 12e⟩
    ⟨deallocate memory 10h⟩

    RESULTIS 0
}
```

2.2 Determine the Number of Digits

The program expects the desired number of decimal digits to the right of the decimal point on the command line. If we don't get a number, or if the number is nonpositive, we print an error message and exit.

8b ⟨local variables 8b⟩≡ (8a) 10a▷

```
LET argv = VEC 10
LET digits = ?
```

8c ⟨determine number of digits 8c⟩≡ (8a)

```
UNLESS rdargs("DIGITS/A", argv, 10) DO ⟨write usage and exit 9⟩
digits := str2numb(argv!0)
UNLESS 0 < digits DO ⟨write usage and exit 9⟩
```

9 ⟨*write usage and exit* 9⟩≡ (8c)

```
{
    writes("usage: pi #digits*n")
    RESULTIS 1
}
```

2.3 Compute π

Following historical precedent, we'll use Störmer's formula

$$\pi = 24 \arctan(1/8) + 8 \arctan(1/57) + 4 \arctan(1/239)$$

and evaluate the arctangents by the Gregory series

$$\arctan x = \frac{x}{1} - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \cdots$$

This was the scheme used by Shanks and Wrench to break the 100,000 digit barrier in 1961.

2.3.1 Algorithm

A simple change of variable gives

$$\arctan \frac{1}{y} = \frac{1}{1y} - \frac{1}{3y^3} + \frac{1}{5y^5} - \frac{1}{7y^7} + \frac{1}{9y^9} - \frac{1}{11y^{11}} + \cdots \tag{17}$$

Now, if we let $t_0 = 1/y$ and $s_0 = t_0$, we get $t_{i+1} = -t_i/y^2$ and $s_{i+1} = s_i + t_{i+1}/(2i+1)$. Obviously, both the ts and ss must be retained as extended-precision numbers.

Given t_i, we can compute t_{i+1} with one division of an extended-precision number (t_i) by a single word (y^2). Likewise, given s_i and t_{i+1}, we can compute s_{i+1} with a division of one extended-precision number (t_{i+1}) by a single word ($2i + i$) and then an addition of two extended-precision numbers (s_i and $t_{i+1}/(2i+1)$).

We also observe that we can compute $n \arctan x$ by replacing the 1s in the numerators of (17) with n.

2.3.2 Representation

That still leaves us with the problem of just how we're going to represent an extended-precision number, since there's no way we can pack thousands or even millions of digits into a single machine word. The obvious solution is a vector. So, how big a vector do we need?

We know a word is at least 32 bits, which is about nine-and-a-half decimal digits. In order to simply the output (Section 2.4), we'll limit ourselves to 8 decimal digits per word, or two decimal digits per byte. The first byte of the vector will hold the integer portion (we know this will fit into a single byte) and subsequent bytes will hold the fractional portion, two digits per byte. Thus, a vector of size $1 + n/8$ will hold a n-digit number with 1 to 8 additional decimal digits.

10a ⟨*local variables* 8b⟩+≡ (8a) ◁8b 10c▷
```
LET bytes, words = ?, ?
```

10b ⟨*allocate memory* 10b⟩≡ (8a) 10e▷
```
bytes, words := 1 + digits / 2, 1 + digits / 8
```

We need three such vectors, one for the running total and two temporaries.

10c ⟨*local variables* 8b⟩+≡ (8a) ◁10a 13a▷
```
LET pi, temp1, temp2 = ?, ?, ?
```

Bear in mind that the first word in the vector returned by `getvec` is reserved for heap management. Thus, we need to allocate one additional word. We'll then advance the pointer by that one word so we don't have to worry about it.

10d ⟨*local procedures* 10d⟩≡ (8a) 11a▷
```
LET allocate(w) = 1 + getvec(w + 1)
```

10e ⟨*allocate memory* 10b⟩+≡ (8a) ◁10b 10f▷
```
pi, temp1, temp2 := allocate(words), allocate(words), allocate(words)
```

If there's insufficient memory, we can't go any further.

10f ⟨*allocate memory* 10b⟩+≡ (8a) ◁10e 10g▷
```
UNLESS pi & temp1 & temp2 DO {
    writes("pi: insufficent memory*n")
    RESULTIS 2
}
```

The running total needs to be initialized to zero.

10g ⟨*allocate memory* 10b⟩+≡ (8a) ◁10f
```
FOR i = 0 TO bytes DO pi%i := 0
```

We should free the memory when we're done, remembering that we advanced the pointers by one word each.

10h ⟨*deallocate memory* 10h⟩≡ (8a)
```
freevec(pi - 1)
freevec(temp1 - 1)
freevec(temp2 - 1)
```

2.3.3 Arctangent

Given the representation of an extended-precision number, we can write a procedure to compute the arctangent. Actually, we will compute $factor \times \arctan \frac{1}{y}$.

11a ⟨*local procedures* 10d⟩+≡ (8a) ◁10d 12b▷
```
LET arctangent(sum, term, temp, factor, y, len) BE {
    LET j, y2 = 1, y * y
    LET acc, carry = ?, ?

    ⟨compute arctangent 11b⟩
}
```

We first compute the first term in the series ($factor/y$) and add it to the running total. (We can skip the division by 1.)

11b ⟨*compute arctangent* 11b⟩≡ (11a) 11d▷
```
term%0 := factor
FOR i = 1 TO len DO term%i := 0
divide(term, term, y, len)
FOR i = 0 TO len DO temp%i := term%i
⟨add to total 11c⟩
```

To add two extended-precision numbers, we start at the bottom and work our way up just as we learned to do it by hand with paper and pencil. (Except for those of you who do it the way Johann Dase did it.)

11c ⟨*add to total* 11c⟩≡ (11)
```
carry := 0
FOR i = len TO 0 BY -1 DO
    acc, sum%i, carry := sum%i + temp%i + carry, acc MOD 100, acc / 100
```

Now we sum the terms until the series converges. We unroll the loop once, dealing with the alternating signs by first subtracting from and then adding to the running total.

11d ⟨*compute arctangent* 11b⟩+≡ (11a) ◁11b
```
{
    ⟨compute next term 11e⟩
    ⟨subtract from total 12a⟩
    ⟨compute next term 11e⟩
    ⟨add to total 11c⟩
} REPEATUNTIL iszero(temp, len)
```

As we've seen above, we can compute the next term by dividing the previous partial term by y^2 and then dividing the result by $2i + 1$.

11e ⟨*compute next term* 11e⟩≡ (11d)
```
divide(term, term, y2, len)
j := j + 2
divide(term, temp, j, len)
```

We've already seen how to add two extended-precision numbers. Subtracting one from another is not very different.

12a ⟨*subtract from total* 12a⟩≡ (11d)
```
carry := 0
FOR i = len TO 0 BY -1 DO
    acc, sum%i, carry := 100 + sum%i - temp%i - carry,
                         acc MOD 100, 1 - acc / 100
```

Since the Gregory-Leibnitz series is absolutely convergent when $|x| \leq 1$, we can consider it converged when the latest term added to the series is 0.

12b ⟨*local procedures* 10d⟩+≡ (8a) ◁11a 12c▷
```
AND iszero(v, l) = VALOF {
    FOR i = 0 TO l IF v%i RESULTIS FALSE
    RESULTIS TRUE
}
```

Division of an extended-precision number by a single word doesn't take much more work than adding two extended-precision numbers. Of course, we'll be in trouble if the term $2i + 1$ ever becomes too big to fit in a single word...

12c ⟨*local procedures* 10d⟩+≡ (8a) ◁12b
```
AND divide(src, dst, d, l) BE {
    LET rem, n = 0, ?

    FOR i = 0 TO l DO
        n, rem, dst%i := (src%i + 100 * rem), n MOD d, n / d
}
```

2.3.4 π

Given the arctangent procedure and its helpers, computing π is a simple matter of computing the three arctangent terms.

12d ⟨*compute* π 12d⟩≡ (8a)
```
arctangent(pi, temp1, temp2, 24,   8, bytes)
arctangent(pi, temp1, temp2,  8,  57, bytes)
arctangent(pi, temp1, temp2,  4, 239, bytes)
```

2.4 Print π

Computing π doesn't do us much good if we can't output the result. And, for readability, we'll do it in decimal.

2.4.1 Integer Portion

We first print out the integer portion.

12e ⟨*print* π 12e⟩≡ (8a) 13b▷
```
writef("pi = %i1.+*n", pi%0)
```

2.4.2 Fractional Portion

We print out the fractional portion in blocks of ten digits, five blocks per line. We'll also put a blank line between blocks of one thousand digits.

13a ⟨*local variables* 8b⟩+≡ (8a) ◁10c

```
LET p = 0
```

13b ⟨*print* π 12e⟩+≡ (8a) ◁12e 13c▷

```
FOR i = 1 TO digits DO {
    TEST i MOD 2 THEN {
        p := p + 1
        writef("%i1", pi%p / 10)
    } ELSE
        writef("%i1", pi%p MOD 10)
    UNLESS i MOD 10 DO {
        wrch(i MOD 50 -> ' ', '*n')
        UNLESS i MOD 1000 DO
            wrch('*n')
    }
}
```

If we haven't printed a complete line, we need to finish off the last one.

13c ⟨*print* π 12e⟩+≡ (8a) ◁13b

```
IF digits MOD 50 DO wrch('*n')
```

2.5 Conclusion

So, how does our program stack up with other methods through the ages? Let's take a look.

2.5.1 Results

The first criterion must be correctness. And our program does indeed produce the correct results. Here are the first thousand digits.

```
pi = 3.+
1415926535 8979323846 2643383279 5028841971 6939937510
5820974944 5923078164 0628620899 8628034825 3421170679
8214808651 3282306647 0938446095 5058223172 5359408128
4811174502 8410270193 8521105559 6446229489 5493038196
4428810975 6659334461 2847564823 3786783165 2712019091
4564856692 3460348610 4543266482 1339360726 0249141273
7245870066 0631558817 4881520920 9628292540 9171536436
7892590360 0113305305 4882046652 1384146951 9415116094
3305727036 5759591953 0921861173 8193261179 3105118548
0744623799 6274956735 1885752724 8912279381 8301194912
9833673362 4406566430 8602139494 6395224737 1907021798
6094370277 0539217176 2931767523 8467481846 7669405132
0005681271 4526356082 7785771342 7577896091 7363717872
1468440901 2249534301 4654958537 1050792279 6892589235
4201995611 2129021960 8640344181 5981362977 4771309960
5187072113 4999999837 2978049951 0597317328 1609631859
5024459455 3469083026 4252230825 3344685035 2619311881
7101000313 7838752886 5875332083 8142061717 7669147303
5982534904 2875546873 1159562863 8823537875 9375195778
1857780532 1712268066 1300192787 6611195909 2164201989
```

2.5.2 Speed

The next thing we might consider is speed. Here we see the penalty for running under an interpreter such as cintsys. On the author's machine, this program requires just under one second to duplicate ENIAC's 2,067 digits, while a C++ program can do in about a tenth of a second. A native code version of the BCPL compiler should be able to close the gap, however.

On the author's 3.2 GHz Pentium4 laptop computer running under cintsys, this program computes 1,000 digits in about a fifth of a second, 10,000 digits in just over twenty seconds, and 100,000 digits in somewhat over half an hour. Extrapolating the quadratic runtime, we can estimate the better part of a day for half a million digits and between two and three days for one million. Obviously, we won't be challenging Kanada for the world record any time soon, but the hardware/software platform is roughly comparable to the mainframe systems used at the Commisariat à l'Energie Atomique in Paris in the 1960s. Not too shabby for a six-pound machine that can run on batteries (though the batteries would run out before a million digits could be computed).

A Index of Code Fragments

Underlined entries are to the definition of the Code Fragment.

B Index of Identifiers

Underlined entries mark the point of definition.