–contents of X Examples–

X Examples

Example – primes

Suppose you wish to test whether an integer is prime. A simple approach is to examine the remainder after division by each feasible divisor. One can quit after trying everything up to the square root of the candidate value. Since fixed size 2's complement arithmetic is used, this approach will eventually fail for large integers.

```
n := PrimeCandidate;                    ` input integer
trial := 2;                             ` smallest possible
do trial*trial < n & n//trial ~= 0 ?
  trial := trial + 1                    ` keep trying
od;
NumberIsPrime := trial*trial > n      ` output
```

In this case the input/output might appear as:

```
PrimeCandidate := 97;
NumberIsPrime := true;
```

The publication form of the prime program is

n ← PrimeCandidate;   ` *input integer*
trial ← 2;   ` *smallest possible*
do trial×trial < n ∧ n//trial ≠ 0 ⇒ trial ← trial+1   ` *keep trying*
od;
NumberIsPrime ← trial×trial > n   ` *output*

Example – π

A simple (and inefficient) way to compute π is to sum the series
$$4 * (1/1 - 1/3 + 1/5 - 1/7 \ldots)$$
It turns out that the average of the last two partial sums is more accurate than the last sum by itself. On a machine using 32-bit IEEE floating point, one should not expect more than about 7 decimal digits accuracy.

```
old, sgn, step, lim := 0.0, 1.0, 1, 1000;
do step < lim ?
  new := old+sgn/i2r(step);
  step := step+2;
  sgn := -sgn
  pi := 2.0*(old+new);              ` result
  old := new;
od
```

The (perhaps surprisingly inaccurate) output is

```
pi := 3.1415913;
```

Given suitable choices in the typesetting options, the typeset version might look like:

old, sgn, step, lim ← 0.0, 1.0, 1, 1000;
do step < lim ⇒
  new ← old+sgn/i2r(step);
  step ← step+2;
  sgn ← −sgn;
  π ← 2.0×(old+new);   ` result
  old ← new;
od

Example − e

The base of natural logarithms can be expressed as a continued fraction:
  2 1 2 1 1 4 1 1 6 1 1....
Because the pattern is regular after the first two items 2 1, the rest can be generated on the fly. Here is a program that does it. [1]

```
t := tol+0.0;                 ` input
i := 2.0;                     ` start at 2 1 1 4 1 1 ...
ao, a := 2.0, 3.0;            ` numerators
bo, b := 1.0, 1.0;           ` denominators
diff := ao/bo - a/b;
do diff*diff < t*t ?
  ao, a := a, ao+a*i;         ` i
  bo, b := b, bo+b*i;
  ao, a := a, ao+a;           ` 1
  bo, b := b, bo+b;
  ao, a := a, ao+a;           ` 1
  bo, b := b, bo+b;
  i := i+2.0
```

---

[1] See Knuth, The Art of Computer Programming, volume 2 for more information.

```
        diff := ao/bo - a/b;
od;

e  := a/b                          ` output
```