Figure 1. An Image for Animation Tests

Figure 1. A Drawdown

Answers to Quiz on Sequences

See Icon Analyst 54 (page 16) for the quiz.

In the answers for problems 1 though 7, we've put some of complicated expressions on separate lines to improve readability. Note that they all are mutual evaluation expressions.

1. (i := seq(), if i == reverse(i) then i)

2.
```
(
  k:= &null, m := 0, i := seq(),
  if /k then k := i else 1,
  if k ~= i then
    (
    m +:= 1,
    if m % 2 = 1 then i else 1(.k, k := i)
    )
  )
```

Comment: This solution seems unnecessarily involuted and complicated, but we couldn't find a better one. See the quiz on the next page, where we pose this problem again for solution using a PDCO.

3. (i := seq(), 1 to i)

4. (i := seq(), (1 to i) | (i – 1 to 1 by –1))

5. (i := seq(), j := 0, (every j +:= !i) | j)

6.
```
(
    i := seq(),
    repeat {
      j := 0
      every j +:= !i
      if *j = 1 then break j
      i := j
      }
    )
```

7.

(a) 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6 …

(b) 1, 1, 4, 1, 4, 9, 1, 4, 9, 16, 1, 4, 9, 16, 25, 1, 4, 9, 16, 25, 36, 1, 4, 9, 16, 25, 36, 49, 1, 4, …

(c) 1, 1, 2, 3, 4, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, …

(d) 1, 1, 4, 9, 16, 1, 4, 9, 16, 25, 36, 49, 64, 81, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, …

(e) 1, 1, 1, 1, 1, 2, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 1, 2, 1, 2, 3, 1, 1, 2, 1, 2, 3, 1, 1, 1, 2, 1, 2, 3, …

(f) 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, …

(g) 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …

(h) 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, …

Note: The second generator in the alternation posed is a red herring; it never gets evaluated because the first generator produces an endless sequence.

(i) –1, 1, –1, 1, –1, 1, –1, 1, –1, 1, –1, 1, –1, 1, …

(j) There was an extra parenthesis in the expression given. It should have been (5 * seq()) % 9, for which the sequence is

5, 1, 6, 2, 7, 3, 8, 4, 0, 5, 1, 6, 2, 7, 3, 8, 4, 0, 5, 1, 6, 2, 7, 3, 8, 4, 0, 5, 1, 6, 2, 7, 3, 8, 4, 0, …

8.

(a) !seq()

(b) !seq() % 7

(c) seq() \ seq(1,2)

(d) (i := seq(), i ^ i)

(e) We accidentally deleted a digit in the fourth term for the sequence given. The sequence should have been

3, 81, 19683, 43046721, 847288609443, 150094635296999121, …

for which a solution is 3 ^ (seq() ^ 2).

Downloading Icon Material

Implementations of Icon are available for downloading via FTP:

ftp.cs.arizona.edu (cd /icon)

Quiz — Programmer-Defined Control Operations

1. Write PDCOs as follows:

(a) ExchangePDCO{}, which exchanges the order of successive terms in a sequence. For example, ExhangePDCO{seq()} should produce

2, 1, 4, 3, 6, 5, …

(b) CumulativePDCO{}, which produces the cumulative sum of a sequence of integers. For example, CumulativePDCO{seq()} should produce

1, 3, 6, 10, 15, …

(c) NonintegerPDCO{}, which filters out non-integer values in a sequence.

(d) ModnPDCO{}, which produces the remainder of each term in an integer sequence divided by its position. For example,

```
ModnPDCO{
  InterleavePDCO{seq(), seq() ^ 2, seq() ^ 3}
  }
```

should produce

0, 1, 1, 2, 4, 2, 3, 1, 0, …

(See page 11.)

Which of the PDCOs above have potential problems with infinite sequences?

2. Show the sequences these expressions produce and describe them in words (see pages 6 and 10-13 for a description of the procedures used).

(a) BinopPDCO{"+", seq(), fibseq()}

(b) BinopPDCO{|!"+–", primeseq(), fibseq()}

(c) ReplPDCO{seq(), seq()}

(d) DeltaPDCO{primeseq()}

(e) OddEvenPDCO{fibseq()}

(f) InterleavePDCO{fibseq(), primeseq()} % 8

(g) InterleavePDCO{fibseq(), primeseq() % 8}

(h) UnopPDCO{"*", fibseq()}

(i) UnopPDCO{"!", seq()}

3. Figure out these sequences, describe them in words, and write expressions that produce them. All can be produced by the procedures described on pages 6 and 10-13.

(a) 1, 1, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, …

(b) 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10, 10, 11, 11, …

(c) 2, 5, 3, 6, 5, 8, 7, 10, 11, 14, 13, 16, 17, 20, 19, 22, 23, 26, 29, 32, 31, 34, 37, 40, 41, 44, …

(d) 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 14, 17, 18, 19, 20, 23, 24, 29, 30, 31, 32, 37, 38, 41, 42, 43, 44, …

4. Describe in words what these PDCOs do.

(a)

```
procedure Puzzle1PDCO(L)
  local x

  while x := @?L do suspend x

end
```

(b)

```
procedure Puzzle2PDCO(L)
  local i

  suspend @L[1]

  repeat {
    i := @L[2] | fail
    every 1 to i do
      @L[1] | fail
    suspend @L[1]
    }

end
```

(b)

```
procedure Puzzle3PDCO(L)
  local i, j

  j := @L[1] | fail

  while i := @L[1] do {
    if i > j then suspend j to i – 1
    else if i < j then suspend j to i + 1 by –1
    else suspend j
    j := i
    }

  suspend j

end
```

Dobby Looms and Liftplans

Keep in mind that a loom like a large dog is more afraid of you than you are of it. — Allen Fannin

The kind of loom we've used as a model in previous articles on weaving [1, 2] has foot powered treadles that raise shafts to make sheds through which successive weft threads pass (*picks*). A treadle can be tied up to several shafts, but only one treadle is pressed for each pick. Some looms prevent more than one treadle from being pressed, but even for looms that do not, since human beings only have two feet, it's not practical to press more than two treadles at a time. As far as we know, it's not done.

The reason that multiple treadling is useful is that it allows more complex weaves to be produced by providing a greater variety of sheds with a given number of shafts and treadles.

Dobbies

The solution to the multiple-treadling problem is a *dobby*, which allows any combination of shafts to be raised using only a single treadle. Dobbies are mounted atop conventional floor looms. They are controlled by a mechanism that originally rotated a belt containing rows of pegs that determined which shafts were lifted. The "peg plan" was set up in advance for a particular weave. One loop around the belt produced one weft repeat.

Modern dobbies have electrical controllers to determine what shafts to raise (although the weaver provides the foot power to raise the shafts). Most dobby devices now are driven by weaving programs running on personal computers.

Figure 1 shows a loom equipped with a simple dobby but without the related paraphernalia. The wires from the dobby are connected to the shafts, and different patterns of wires are raised to produce different sheds.

Figures 2 and 3 show a loom with a more sophisticated dobby and its controlling devices.

Figure 1. Schact Floor Loom with Dobby

Figure 2. AVL Compu-Dobby Loom