CSc 372

Comparative Programming Languages

36 : Icon — Exercises

Christian Collberg

collberg+372@gmail.com

Department of Computer Science

University of Arizona

Expressions

- What are the results of these expressions?

1. `(1 to 5) | (5 to 1 by -1`
2. `(1 < 0) | (0 = 1)`
3. `(1 < 0) | ( ~= 1)`
4. `write((3 | 7 | 13) > 10)`

Expressions

- Give the values of the expressions below:

```
1. every writes(![1, 3, 5], " ")

2. while writes(![1, 3, 5], " ")

3. every writes(17 % (2 to 5), " ")

4. writes(!"hello", " ")

5. every writes(!"hello", " ")

6. while writes(!"hello", " ")

7. writes(0 < 18
```

Generators

```
procedure Gen()
   every i := 100 to 500 by 100 do
     suspend i - 1
     every i := !"abc" do
     suspend i
end
```

- Given the procedure above, give the values of the expressions below:

```
1. every writes(Gen(), " ")

2. writes(*Gen(), " ")

3. every writes(*Gen(), " ")
```

Generators

```
procedure Gen()
   lst := [11, 12, 13, 14, 15, 16, 17]
   while *lst > 1 do {
      suspend(pull(lst))
      suspend(pop(lst))
   }
   suspend 58
end
```

- Give the values of the expressions below:

```
1. every writes(Gen(), " ")

2. while writes(Gen(), " ")

3. every writes(!Gen(), " ")

4. every writes(*Gen(), " ")
```

Exercise

- Set a variable digits to a string consisting of the result of taking 7 mod the integers 1 through 80.
- In other words, the first digit should be 1 % 7, the second should be 2 % 7, the third should be 3 % 7, and so on up to the 80th digit which should be 80 % 7.

Exercise

- Given a variable list1 that contains a list of integers, set a variable list2 to the squares of the integers in list1.
- For example, if list1 stores [1, 8, -3, 7], then list2 should be set to [1, 64, 9, 49]

Exercise

- Given a variable list3 that contains a list of integers, set variables oddProduct and evenProduct to the product, respectively, of the odd numbers in the list and the even numbers in the list.
- For example, if the list stores [3, 4, 8, 7, 5], then oddProduct should be set to 105 (3 * 7 * 5) and evenProduct should be set to 32 (4 * 8).
- Assume that list3 contains at least one even and at least one odd

Exercise

- Given a variable n that stores a positive integer, set a variable result to a string containing a random n-digit binary number (i.e., a number composed entirely of the digits 0 and 1 without any leading zeros).
- Your code should produce any of the n-digit binary numbers with equal probability

Exercise

- Set a variable odds to a list containing all 2-digit odd numbers in reverse order (99, 97, ..., 11).

Exercise

- Given a string variable called stuff, set a variable stuff2 to the string obtained by eliminating all spaces from stuff.
- E.g., if stuff is `"hi there, how are you?"` then set stuff2 to `"hithere,howareyou?"`

Exercise

- Given a variable list1 that stores a nonempty list, set a variable list2 to the list obtained by eliminating every other value from list1 starting with the first.
- For example, if list1 stores `[1, 2, 3, "abc", "def", 19]`, then list2 should be set to `[2, "abc", 19]`.

Exercise

- Set a string variable result to a comma-separated list of the integers 1 through 50 enclosed in parentheses.
- In other words, the string should begin with `"(1, 2, 3, 4, 5"` and should end with `"49, 50)"`.

Anagrams

- Write an Icon procedure called anagram that takes two strings as arguments and that determines whether the strings are anagrams of each other. Two strings are considered anagrams if they are composed of the same characters.
- For example, “spear”, “pears", “reaps”, “parse” and “spare” are all anagrams of each other.
- One simple approach is to compare the result of sorting the letters of each word.
- Remember that Icon’s sort procedure can be used only for tables and lists, not for strings.

Evens

- Write an Icon procedure evens(n) that generates the first n even numbers starting with 2.
- For example, the following code:

```
every i := evens(10) do
    writes(i, " ") write()
```

would write the first 10 evens to output: 2 4 6 8 10 12 14 16 18 20

Sort Lines

- Write a complete Icon program that reads from standard input, writing the same text to standard output with the lines in sorted order.
- Example:

```
phil                      george
sally                     harry
jane                      jane
george                    jorge
martha         ⇒          martha
jorge                     mitch
mitch                     phil
harry                     sally
stewart                   stewart
```

Position

- Write an Icon procedure called position that takes two lists of numbers as arguments and that returns the position of the first list in the second list. The first list might appear multiple times in the second list, so you should write this as a generator that returns each starting position of the first list in the second.
- For example, given the call:

  ```
  position([1,2], [1,2,3,4,2,1,1,8,1,2,3,1,2])
  ```

  the procedure should return a position of 1 indicating that the first list appears starting at position 1 in the second list. If called again as a generator, it should next return 9 because the first list also appears starting at position 9. Called again as a generator it should return 12 because the list also appears starting at position 12. After that it should fail if called as a generator.
- If the first list does not appear in the second list, the procedure should fail. Assume that both lists are nonempty and contain just numbers.

Flip

- Write an Icon procedure flip that takes a list as an argument and that returns the list obtained by flipping the order of successive pairs of values in the original list.
- For example:

```
flip([1, 3, 17, 4, 9, 12])
```

  should return [3, 1, 4, 17, 12, 9].
- If the list has odd length, then the final value from the original list should appear in its original position in the new list. For example, flip([1, 2, 3]) should return [2, 1, 3].
- You are to return a new list and are not allowed to change the original list.

Rotations

- Write an Icon procedure rotations that takes a list as an argument and that generates a series of rotations of the list as sequential answers. Your procedure should be written as a generator which produces each of these answers.
- For a nonempty list of length n, your procedure should generate n different answers. The first answer should have the same values as the original list in the same order. The second answer should be rotated once, with the value at the front of the list rotated to the back of the list. The third answer should be rotated twice, with the first two values rotated to the back of the list. And so on.

Rotations (cont)

- For example, given the list [1, 2, 3, 4, 5, 6, 7, 8], your procedure should generate 8 different answers:

```
[1, 2, 3, 4, 5, 6, 7, 8]
[2, 3, 4, 5, 6, 7, 8, 1]
[3, 4, 5, 6, 7, 8, 1, 2]
[4, 5, 6, 7, 8, 1, 2, 3]
[5, 6, 7, 8, 1, 2, 3, 4]
[6, 7, 8, 1, 2, 3, 4, 5]
[7, 8, 1, 2, 3, 4, 5, 6]
[8, 1, 2, 3, 4, 5, 6, 7]
```

- Your procedure should create a copy of the list to return these rotations. The original list should remain unchanged. If passed an empty list, your procedure should produce an empty list as its one and only answer.

Acknowledgments

- These problems were taken from tests produced by Stuart Reges.