CSc 372

Comparative Programming Languages

38 : Summary

Christian Collberg

collberg+372@gmail.com

Department of Computer Science

University of Arizona

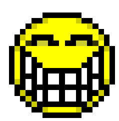
CSc 372 — What Was *That* All About?!?!

What Have We Learned?

- Three languages!
- More importantly, we've learned to learn new languages! You will be doing this many times during your careers.
- We've learned to quickly understand new formalisms. Formalisms aren't just programming languages, but also include formal specifications of programs, XML specifications, game scripts, etc.
- We've learned to think about problem-solving in new ways. Next time you see a problem that cries out for backtracking you will think of Prolog and Icon!

3 Languages

You Are ~~Not~~ Supposed to Understand This Lecture!!!



Haskell

Sample interaction:

```
?   commaint "1234567"
    1,234,567
```

`commaint` in Haskell:

```
commaint = reverse .  foldr1 (\x y->x++","++y) .
           group 3 .  reverse
           where group n = takeWhile (not.null) .
         map (take n).iterate (drop n)
```

Prolog

A coloring is OK iff

1. The color of Region 1 $\neq$ the color of Region 2, and
2. The color of Region 1 $\neq$ the color of Region 3,...

```
color(R1, R2, R3, R4, R5, R6) :-
   diff(R1, R2), diff(R1, R3), diff(R1, R5), diff(R1, R6),
   diff(R2, R3), diff(R2, R4), diff(R2, R5), diff(R2, R6),
   diff(R3, R4), diff(R3, R6), diff(R5, R6).

diff(red,blue).  diff(red,green).  diff(red,yellow).
diff(blue,red).  diff(blue,green).  diff(blue,yellow).
diff(green,red).  diff(green,blue).  diff(green,yellow).
diff(yellow, red).diff(yellow,blue).  diff(yellow,green).
```

Icon

```
# Convert all occurrences of A,E,H,I,O, U,W,Y in other
# positions to "." # Assign the following numbers to
# the remaining letters after the first:
# B,F,P,V => 1                L => 4
# C,G,J,K,Q,S,X,Z => 2        M,N => 5
# D,T => 3                    R => 6
procedure soundex(name)
   local first, c, i
   # Convert to uppercase.
   name := map(name, string(&lcase),string(&ucase))

   # Retain the first letter of the name
   first := name[1]
   name := map(name, "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
                     ".123.12..22455.12623.1.2.2")
```

Icon. . .

```
    # If two or more letters with the same
    # code  were adjacent in the original name,
    # omit all but the first
    every c := !"123456" do
        while i := find(c||c,name) do
            name[i+:2] := c
    name[1] := first

    # Now delete our place holder ('.')
    while i := upto('.',name) do name[i] := ""
    return left(name,4,"0")
end

procedure main(args)
    write(args[1] || " ==> " || soundex(args[1]))
end
```

Final Exam

The Final Exam

- The final is comprehensive.
- Typical questions will include:
  - What does this expression compute?
  - What does this mysterious function/procedure/predicate compute?
  - Write a function/procedure/predicate that . . .
  - Give a short definition of the term . . . and provide a short, illustrating, example.
- **The final is Wednesday, December 14, 14:00-16:00, 2005 in GLD-S 906**. See `http://www.registrar.arizona.edu/schedule054/exams/mwf.htm`.
- Below are the relevant problems from last year's final exam.

Problem 1

Assume that the following procedure has been defined:

```
procedure Gen()
    local T,i

    T := table(0)
    T[1] := "foo"
    T[2] := "bar"
    T[3] := "foobar"
    T[4] := "baz"

    i := 1
    while i <= *T do {
        suspend T[i]
        i +:= 1
    }
end
```

Problem 1...

For each Icon expression below, indicate what output is produced by the expression. If the output is empty, list nothing. If the output is infinite, indicate enough of the output to show the pattern and then use the ellipsis ("...") to indicate that it continues forever.

1. `every i := Gen() do writes(i, " ")`
2. `every i := *Gen() do writes(i, " ")`
3. `every i := !Gen() do writes(i, " ")`
4. `every i := *!Gen() do writes(i, " ")`
5. `write(Gen())`

Problem 1...

6. `while i := Gen() do writes(i, " ")`

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

8. `while (*(i := Gen())>4) do writes(i," ")`

9. `every(writes(1 to 10 by 2, " "))`

10.
```
T := table(0)
T[1] := "foo"
T[2] := "bar"
T[3] := "foobar"
T[4] := ["foo","bar"]
L := ["foo",set(T[4]), "bar", T, "baz"]
every (write(*!L))
```

Problem 2

For each language we have studied in this class, write a function/predicate/procedure that takes a list of integers as input and returns its sum. In this problem, *style* as well as *correctness* counts: for each language you should use the most idiomatic (most natural) way to write the code. I.e., for Haskell you might want to use composition of higher-order functions rather than recursion, for Icon generators rather than while loops, etc. Remember to give type signatures where necessary.

1. In Prolog, `sum` is invoked like this:

```
| ?- sum([1,2,3,4,5],L).
L = 15
```

2. In Haskell, `sum` is called like this:

```
> sum [1,2,3,4,5]
15
```

Problem 2...

3. In Icon, the statement

```
write(sum([1,2,3,4,5]))
```

   should print `15`.

Problem 3

1. Given this Prolog predicate definition

```
mystery(L, B) :-
    member(X, L),
    append(A,[X],L),
    append(B,C,A),
    length(B,BL),
    length(C,CL),
    BL > CL.
```

what does the query

```
| ?- mystery([1,2,3,4,5],C), write(C), nl, fail.
```

print?

Problem 3...

2. Given these Haskell function definitions

```
mystery :: [a] -> [[a]]
mystery xs = [take n xs,drop n xs]
             where n  = h  xs


h :: [a] -> Int
h [] = 0
h [_] = 0
h (_:_:xs) = 1 + h xs
```

what does the expression

```
mystery [1,2,3,4,5]
```

return?

Problem 4

1. Write an Icon generator `flatten(L)` that produces the elements of a nested list. For example, the program

```
every i := flatten([1,[2,3],[4,[5]],6]) do
  write(i)
```

should print out

```
1
2
3
4
5
6
```

Problem 4. . .

2. Write a procedure `strip` that uses string scanning primitives to remove all non-letters from a string. For example, the statement

   ```
   write(strip("h5i the3re!"))
   ```

   should print the string

   ```
   hithere
   ```

Problem 5

1. What is *referential transparency*? Illustrate with an Icon procedure and a Haskell function.

2. Haskell is a *lazy* language. What does this mean?

3. What is an *accumulator* pair? What is it good for? Illustrate with a Prolog or Haskell example.

[20]

Thanks for being a great class!

GOOD LUCK ON THE FINAL!!!

And have a wonderful break!