CSc 372

Comparative Programming Languages

28 : Icon — Introduction

Christian Collberg

collberg+372@gmail.com

Department of Computer Science

University of Arizona

Introduction

The Icon Language

- Icon is a prototyping language that traces its ancestry from Pascal and SNOBOL.
- Icon is dynamically typed. It has generators, string manipulation functions, coroutines, structured data types (lists, tables, and sets), garbage collection, and built-in graphics support.
- Pick up implementations for Unix, Mac, PC, etc from `ftp.cs.arizona.edu`.
- With the implementation comes a huge library of useful routines and programs.
- Icon programs are usually interpreted, but there is also a compiler that translates to `C`.

History

- Defined by Ralph Griswold, Prof. Emeritus at the University of Arizona.
- Derived from SNOBOL (also by Griswold) and SL5 (Griswold and Dave Hansen).
- Name comes from Iconoclast.

```
"The Collaborative International Dictionary of English v.0.48"
Iconoclast I*con"o*clast, n. Gr. e'ikw`n image + ? to break:
   cf. F. iconoclaste.
   1. A breaker or destroyer of images or idols; a determined
      enemy of idol worship.

   2. One who exposes or destroys impositions or shams; one who
      attacks cherished beliefs; a radical.
```

Running Icon

Icon Modules

- An Icon program consists of a number of procedures declared in one or more modules. Modules are separately compiled.
- Each program must have a procedure `main` that will be called first when the program is started.

A.icn

```
procedure X()
  ...
end

procedure  Y()
  ...
end
```

B.icn

```
procedure Z()
  ...
end
```

M.icn

```
link A, B

global r, t

procedure main ()
   local s, t
   X()
   Z()
end
```

Compiling Icon Programs

- Set these environment variables:

```
setenv IPATH /usr/local/lib/icon/lib
setenv LPATH /usr/local/lib/icon/include
setenv FPATH /usr/local/lib/icon/bin
```

- To compile an Icon module `M.icn` do `icont -c M.icn`. This generates two files `M.u1` and `M.u2`.
- To link an Icon program (where the `main` procedure is in the module `M.icn`) do `icont M.icn`. This generates an executable file `M`.
- You can pick up additional Icon programs and functions from `/usr/local/lib/icon/lib/bipl` and `/usr/local/lib/icon/lib/gipl`.

Procedure Declarations

- A procedure has five parts: The heading, local declarations, initializations, static declarations, and the procedure body.
- A variable that is declared static survives between procedure invocations.
- Statements in an initial clause are run the first time the procedure is called.

```
global R, T
procedure name (arguments, extra[])
   local x, y, z
   static a, b, c
   initial { ...  }
   <statements>
end
```

Interactive Icon

- Normally we run Icon by saving the program in a file and compiling it to bytecode using `icont`.
- William Mitchell has written a program `ie` (Icon Evaluator) that allows us to try out Icon expressions interactively.
- The source is here:

  http://www.mitchellsoftwareengineering.com/icon/ie.icn

- You can also run it directly on lectura:

```
> setenv IPATH ${IPATH}:/home/cs372/fall03/icon/lib
> /home/cs372/fall03/icon/ie
Icon Evaluator, Version 0.8.1, ? for help
][ 5+7;
   r1 := 12  (integer)
```

Program Layout

- Icon is expression-oriented — every program construct returns a value.
- Expressions can be separated by blank lines or semicolons, or both.
- These are equivalent:

```
write("hi"); write(5)

write("hi");
write(5)

write("hi")
write(5)
```

Icon programmers avoid using semicolons whenever possible.

Program Layout...

- Long lines can be broken after an operator:

```
x := something + something_else *
     something_different
```

Preprocessor

- There is a simple pre-processor that allows you to define constants:

```
$define MaxVal 1000
...
if i > MaxVal then ...
```

Debugging Icon

Debugging Icon

- Bad news: There is no Icon debugger. Good news: You don't need one!
- Since the time for an edit-compile-link is so fast, you can do your debugging using `write` statements.
- `SETENV TRACE=-1` or `&trace:=-1` will trace function calls.

Debugging Icon. . .

- When a runtime error occurs, execution terminates, and a *traceback* (a list of all active procedure calls) is generated:

```
procedure Q(); x:=x+"hello"; end
procedure P(); Q(); end
procedure main(); P(); end
```

$\Downarrow$

```
Run-time error 102
File s.icn; Line 7
numeric expected
Trace back:
   main()
   P() from line 3 in s.icn
   Q() from line 2 in s.icn
   {&null + "hello"} from line 1
```

Debugging Icon...

- `xdump` will display any variable type:

```
link ximage
procedure main()
   x := table(0); x[5]:="c"
   xdump([99,set([3,4]),x])
end
```

$$\Downarrow$$

```
L2 := list(3)
      L2[1] := 99
      L2[2] := S1 := set()
         insert(S1,3)
         insert(S1,4)
      L2[3] := T1 := table(0)
         T1[5] := "c"
```

Introductory Example

Soundex

- When names are communicated by telephone, they are often transcribed incorrectly.
- Soundex is a system of encoding a name that will mitigate the effects of transcription errors.

```
# 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:
```

Soundex...

```
# 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")
```

Soundex...

```
    # 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
```

Soundex...

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

Explanation

```
][ name := "collberg";
][ name := map(name, string(&lcase),string(&ucase));
   r15 := "COLLBERG"  (string)
][ name := map(name, "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
                     ".123.12..22455.12623.1.2.2");
   r16 := "2.441.62"  (string)
][ every c := !"123456" do write(c);
1
2
3
4
5
6
```

Explanation. . .

```
][ while i := find("44",name) do name[i+:2] := "4";
][ write(name);
2.41.62
][ while i := upto('.',name) do name[i] := "";
][ write(name);
24162
][ left("C4162",4,"0");
   r23 := "C416"  (string)
```

Tracing Soundex

`left(s1, i, s2)` shift s1 to the left, append s2:s until position i is reached.

Example

`COLLBERG` ⇒(code) `"2.441.62"` ⇒(remove duplicates) `"2.41.62"` ⇒(restore first) `"C.41.62"` ⇒(delete ".") `"C4162"` ⇒(truncate) `"C416"`

`COLBERG` ⇒(code) `"2.41.62"` ⇒(remove duplicates) `"2.41.62"` ⇒(restore first) `"C.41.62"` ⇒(delete ".") `"C4162"` ⇒(truncate) `"C416"`

Summary

Confused Student Email

<u>Question I</u>

HI Dr. Collberg: Is there any expression in ICON similar to `"&&"` *logical "AND" expression in PASCAL ? Or should I just use:*

```
If (true) then
   if (true) then
```

expr1 & expr2 succeeds (and produces expr2) if both expr1 and expr2 succeed.

Confused Student Email...

Dear Dr. Christian:
I compile and run my program at home on my PC, transfer it to the Unix machine at the department, and then it won't run! What's wrong???
Sincerely,
Confused.

Dear Confused,
The `.u1` and `.u2` files are text files. Be sure to transfer them so that the newline characters are properly converted. Or, transfer the `.icn` file and recompile.

Confused Student Email…

Question VI

While doesn't this work

```
every write(f2, read(f1))
```

while this does:

```
while write(f2, read(f1))
```

`read` is not a generator.

Confused Student Email...

What could cause machcode.icn to lose track of subroutines in other files? My makefile is fine, because at one moment machcode.icn is grabbing external routines correctly then it starts randomly selecting routines to reject (i.e. `&null(variables)`*.) It's even rejected YOUR*

```
Mcode := mcode_Create()
```

the second line of the first procedure!!! And then, without changing a single line of code above it, machcode will accept it again and pick some other external routine to complain about!

Confused Student Email…

Icon doesn't have a module system. In other words, all procedures are global. This is why all (most) my procedures are prefixed by the module name. What could have happened is that you've declared a global variable or record or procedure whose name conflicts with one of my procedures, elsewhere in the compiler. So, try to name all your global procedures/variables/records with unique (i.e. long) names.
Also, make sure that you get the case right; `mcode_Create()` is different from `mcode_create()`.

Readings

- Read `Christopher, Chapter 1`. This is the reference text I will mostly be refering to.
- You can also read the corresponding sections in `Griswold and Griswold`.

References

- The Icon Programming Language, by Griswold and Griswold. Prentice Hall. ISBN 0-13-447889-4.
- The Icon Home Page: http://www.cs.arizona.edu/icon/
- Thomas W Christopher - Icon Programming Language Handbook,

  http://www.tools-of-computing.com/tc/CS/iconprog.pdf
- http://dmoz.org/Computers/Programming/Languages/Icon
- http://www.nmt.edu/tcc/help/lang/icon/homepage.html
- The string-scanning examples were taken from http://www.cs.arizona.edu/icon/intro.htm and

  http://www.nmt.edu/tcc/help/lang/icon.
- Bill Mitchell's Icon Evaluator:

  http://www.mitchellsoftwareengineering.com/icon/ie.icn