CSc 372

Comparative Programming Languages

30 : Icon — Control Structures

Christian Collberg

collberg+372@gmail.com

Department of Computer Science

University of Arizona

Success and Failure

Expressions

- There are fundamental differences in the way Java, C, etc. & Icon statements are executed:
  1. Icon statements are expressions that return values.
  2. Icon expression either succeed or fail.
- Failure doesn't necessarily mean that something has gone wrong, rather, it means that there is no value to return.
- `numeric("pi")` fails because `"pi"` cannot be converted to number.

Success and Failure

$i + j$ Succeeds and returns the value $i + j$.

$i < j$ Succeeds if $i < j$, in which case j is returned. Fails otherwise.

`numeric(x)` Converts x to a number.

`numeric("3.14")` Returns 3.14.

`numeric("pi")` Fails.

- All Icon variables have a special `null` value initially.

Examples

```
][ w := "hello world";
   r2 := "hello world"
][ w[20];
Failure
][ numeric("55");
   r4 := 55  (integer)
][ numeric("pi");
Failure
][ x := 42;
][ x + numeric("10");
   r9 := 52  (integer)
][ x + numeric("pi");
Failure
```

Examples. . .

```
][ x + y;
Run-time error 102
offending value: &null
][ "hi" || w[20];
Failure
```

Comparisons

- Comparisons in Icon succeed or fail:
  - $i < j$ succeeds if i is less than j and fails otherwise.
  - If $i < j$ succeeds then the expression returns j.

```
][ i := 5;
][ j := 6;
][ i < j;
   r16 := 6
][ j < i;
Failure
```

Comparisons. . .

```
][ max := 5;
][ max := max < 6;
   r20 := 6
][ max;
   r21 := 6
][ max := max < 3;
Failure
][ max;
   r23 := 6
][ if min < j < max then
      write("yes") else write("no");
yes
```

- If `min < j` then the expression succeeds and produces `j` which is then compared to `max`.

Expressions

- All Icon expressions return values.

```
][ res := if min < j < max then
       write("yes") else write("no");
][ res;
   r30 := "yes"
][ x := 42;
][ x := 5 + if 1 > 2 then 3;
Failure
][ x;
   r39 := 42
][ x := 5  + if 2 > 1 then 3;
][ x;
   r41 := 8
```

Compund Expressions

- Just like in C and Java, several expressions can be aggregated using the syntax $\{e_1, e_2, \ldots, e_n\}$.
- Each expression is executed in turn.
- The value of the last expression is the result of the compound.
- Failure of one of the expression doesn't make the compund fail.

Compund Expressions — Examples

```
][ {1;2};
   r42 := 2
][ {1>2;3};
   r43 := 3
][ x := if 2>1 then {1; 3+4};
][ x;
   r45 := 7
```

Repetition

while

- The while-expression has the syntax

  ```
  while (expr1) do expr2
  ```

  For as long as expr1 succeeds, expr2 is evaluated.
- The while-expression always fails.

```
i := 0; s := ""
while (i < 10) do
   s ||:= i+:=1 || "."
```

break and next

- break and next behave as in C.

not

- not e succeeds and returns null if e fails.
- not e fails if e succeeds.

```
][ not (1>2);
   r61 := &null
][ not (2>1);
Failure
```

&

- $e_1 \& e_2$ succeeds if both e_1 and e_2 succeed, and the result is the value of e_2.
- e_1 is evaluated first and if it succeeds, e_2 is evaluated.
- If either of e_1 and e_2 fail, $e_1 \& e_2$ fails.

&...

```
][ 1 & 2;
   r63 := 2
][ 1 & 2 & 3;
   r64 := 3
][ 1 & (1 > 2);
Failure
][ write(1) & (1 > 2);
1
Failure
][ (1 > 2) & write(2);
Failure
```

&...

```
procedure main()
   S := ""
   while (line := read()) & (line ~== "end") do
      S ||:= " " || line
   write(" >>> " || S)
end
```

```
> read
hello
world
end
 >>>  hello world
```

Testing for `null`

- `/expr` succeeds if `expr` is `null`, and then returns `null`.
- `\expr` succeeds if `expr` is not `null`, and then returns `expr`.
- Think of “`/e` succeeds if e is `null` because the / falls over, getting no support from e.”

Testing for `null`...

```
][ x := &null;
][ /x;
   r4 := &null
][ \x;
Failure
][ /x := 42;
][ x;
   r7 := 42
][ /x := 10;
Failure
][ x;
   r9 := 42  (integer)
```

Booleans

- There is no boolean type in Icon, but you can use null as False and any non-null value as True.
- if \x & \y then then functions as if x and y then would in other languages.

```
][ x := 1;
][ y := 1;
][ if \x & \y then write(42);
42
][ if \x | \y then write(42);
42
][ if \v | \z  then write(42);
Failure
][ if \z  | \x  then write(42);
42
```

Goal-Directed Evaluation

- Icon supports bounded backtracking within one expression.
- Once e_1 in if e_1 then... has generated a value, no more values are generated.
- Generating one pythagorean triangle:

```
procedure main()
   if i := 1 to 100 & j := 1 to 100 &
      k := 1 to 100 & i^2 + j^2 = k^2 then
      write(i, " ", j, " ", k)
end

> pythagoras
3 4 5
```

until

- until e_1 do e_2 behaves the same as while not (e_1) do e_2.
- If e_1 fails then e_2 gets evaluated.

```
][ x := 1;
][ until x > 10 do x +:= 1;
Failure
][ write(x);
11
```

Fibonacci

```
procedure main()
   local i,j
   i := 1
   j := 1
   until i > 1000000 do {
      write(i)
      i +:= j
      i :=: j
  }
end
```

- x :=: y swaps the two values in x and y.

repeat

- repeat *e* evaluates *e* forever.
- Use break or return to exit the loop.

```
][ i := 1;
][ repeat {i +:= 1; if i > 10 then break;};
][ write(i);
11
```

case

-
  ```
  case e of {
     e1 :  s1
     e2 :  s2
     ...
     default :  s3
  }
  ```
- Similar to repeated if-expression: `if e===e1 then s1 else if e===e2 then s2 else... else s3`. The `default`-part is optional. `e1, e2,...` can be arbitrary expressions of arbitrary type, not just scalar constants as in C's switch statement.
- `===` is the universal equality test. For two numbers it does a numeric test, for two strings, a string test, for other kinds of objects (tables, sets, lists) it checks that the objects are the same object.

Examples

```
][ 5 === 5;
   r4 := 5  (integer)
][ "5" === "5";
   r5 := "5"  (string)
][ [1,2,3] === [1,2,3];
Failure
][ x := [1,2,3];
][ x === x;
   r9 := L1:[1,2,3]  (list)
```

Summary

Readings and References

- Read Christopher, pp 28, 45--52.

Acknowledgments

- Some material on these slides has been modified from William Mitchell's Icon notes:

  http://www.cs.arizona.edu/classes/cs372/fall03/handouts.html.
- Some material on these slides has been modified from Thomas W Christopher's Icon Programming Language Handbook,

  http://www.tools-of-computing.com/tc/CS/iconprog.pdf.