

CSc 372

Comparative Programming Languages

32 : Icon — Procedures

Christian Collberg

collberg+372@gmail.com

Department of Computer Science
University of Arizona

Copyright © 2005 Christian Collberg

—Fall 2005 — 32

[1]

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
```

372 —Fall 2005 — 32

[2]

Procedure Declarations...

```
procedure foo()
  static counter
  initial {counter:=1}
  ...
  counter :=+ 1
end
```

Fall 2005 32

[3]

Parameter Passing

- Parameters are called **by value**.
- This means that actual arguments to a procedure are copied into the formal parameters.
- Any changes to the formals won't affect any of the actuals. This is similar to C and Java.

```
procedure foo(a)
  a := "bye"
end

procedure main()
  local a
  a := "hello"
  foo(a)
  write(a)
end
```

372 —Fall 2005 — 32

[4]

Modules

- Note that Icon doesn't have a real module-system.
- All names (procedure names, record names, global variables) live in the same **name space**.
- You need to make sure that all global names are unique! I usually do this by prefixing all names by the module-name: `mymodule_myproc`.

Formal Parameters

- When you call a procedure you can supply fewer arguments than there are formal parameters:

```
procedure P (f1,f2,f3)
end
```

When calling `P` with `P(a1,a2)` the formal parameter `f1` will take on the value of `a1`, and `f2` will get `a2`. `f3` will become `null`.

Default Parameters

- A common idiom for default parameters:

```
procedure P (f1,f2,f3)
  /f3 := <default value>
end
```
- When calling `P` with `P(a1,a2)`, `f3` will get the default value.
- When calling `P` with `P(a1,a2,a3)`, `f3` will get the value of `a3`.

Arbitrary Length Argument Lists

- Icon supports arbitrary length argument lists:

```
procedure P (f1,f2,f3[])
end
```

When calling `P` with `P(a1,a2,a3,a4,a5)`, the `f3` formal will hold the list `[a3,a4,a5]`.

Procedure Returns

- `return e` returns the value `e`.
- If `e` in `return e` fails, then the procedure call itself fails.

```
procedure less(a)
  return a<10
end
```

```
][ .inc less.icn
][ less(5);
  r1 := 10
][ less(100);
Failure
```

Indirect Procedure Calls

- Procedure names can be constructed at runtime, allowing a powerful form of indirect procedure call.
- Remember to include the directive `invocable all` at the beginning of your module.
- `proc(P)` returns the procedure whose name is the string `P`.

```
P1 := proc("MyProc1")
P2 := proc("MyProc" || "2")
P3 := proc("find")      # Built-ins OK, too.
P4 := proc("*", 2)      # Multiplication has arity 2.
L := [P1, P2, P3, P4]  # A list of procedures.
L[2](45, "X2")        # Calling MyProc2(45, "X2").
```

Readings

- Read `Christopher`, pp. 53--55, 57--58.