CSc 372

Comparative Programming Languages

29 : Icon — Basics

Christian Collberg

collberg+372@gmail.com

Department of Computer Science

University of Arizona

Types and Variables

Types and Variables

- Local variables don't have to be declared, but do it anyway!
- Global variables must be declared.
- An variable that has *not* been declared will automatically be treated as a local variable.
- Icon is dynamically typed. This means that
  - You don't need to declare the types of variables.
  - A variable may contain different types of data at runtime.

```
local X
X := "hello"   # String
X := 5         # Integer
X := 6.7       # Real
```

Types and Variables. . .

- . . .
  - You won't get type errors at compile-time, but you will get them at run-time:

```
procedure main(args)
   t := "hello" + 4.5
end
```

$\Downarrow$

Run-time error 102
File t.icn; Line 6
numeric expected
offending value: "hello"
Trace back:
    main()
    {"hello" + 4.5} from line 2

Examining Types

- `type(V)` will return the *name* (a string) of the type of `V` :

```
record complex(a,b)
t := "hello"
x := type(t)   # x="string"
t := [5,6,7]
x := type(t)   # x="list"
t := complex(4,5)
x := type(t)   # x="complex"
```

Automatic Conversions

- Some data types are automatically converted to the required type. For example, a string (consisting entirely of digits) can be converted into a number, explicitly or implicitly:

```
write(5 + "6")              # implicit
write(5+integer("6"))       # explicit
```

- Icon will try, as much as it can, to satisfy a request.

Examples

```
][ x := 45.9;
   r1 := 45.9  (real)
][ type(x);
   r2 := "real"  (string)
][ "50" / 2;
   r1 := 25  (integer)
][ "50.0"/2.0;
   r2 := 25.0  (real)
][ "50yikes"/2;

Run-time error 102
```

Numbers

Integers

- Integers are arbitrary size.
- Icon has the standard arithmetic operators with the expected precedences: `+,*,-,/,%`. The `^`-operator performs exponentiation.
- Numerical comparison operators: `<,<=,=,>=,>,~=`.
- Bit-functions: `iand,ior,ixor,icom,ishift`.
- `?n` produces an integer between `1` and `n`.

Reals

- Icon uses native real numbers.
- Mathematical functions: `sin,cos,tan,asin,acos,atan,sqrt,exp,log`.
- Mathematical constants: `&pi,&e`.
- `?0` produces a real number between `0.0` and `1.0`.

Examples

```
][ ior(4,6);
   r1 := 6  (integer)
][ ishift(2,3);
   r2 := 16  (integer)
][ 234234324234*2343243243242;
   r3 := 548867997596676357326628  (integer)
][ ?100;
   r4 := 22  (integer)
][ ?100;
   r5 := 42  (integer)
][ ?0;
   r6 := 0.3157951944  (real)
][ ?0;
   r7 := 0.5104401731  (real)
```

Strings

Strings

- Literal strings are given in double quotes: `"hi"`.
- Long strings can be spread over several lines:

```
s := "this is a _
      very long _
      string"
```

- `*s` returns the length of `s`.
- String comparison operators: `<<,<<=,==,>>=,>,~==`.
- String concatenation operator: `||`

Examples

```
][ n := *"hello world";
   r4 := 11  (integer)
][ if "hello" << "world" then
      write("yes") else write("no");
yes
   r5 := "yes"  (string)
][ "hello" || " " || "world";
   r6 := "hello world"  (string)
][ s := "hello";
   r7 := "hello"  (string)
][ s || *s;
   r8 := "hello5"  (string)
```

Augmented Operators

- `a +:= b` means the same as `a := a + b`.
- The same pattern can be used for all binary operators: `a ||:= b` is the same as `a := a || b`.
- `a <:= b` assigns `b` to `a` if `a<b`.

Examples

```
][ s := "hello";
][ s ||:= " world";
   r9 := "hello world"  (string)
][ k := 5;
][ k +:= 10;
   r11 := 15  (integer)
][ m := 5;
][ m <:= 6;
   r13 := 6  (integer)
```

Max — String Comparison

```
procedure main()
   max := read()
   while line := read() do
      max <<:= line
   write(max)
end
```

```
> icont max.icn
> max
10
20
5
30
5
```

Max — Numerical Comparison

```
procedure main()
   max := read()
   while line := read() do
      max <:= line        # Note the difference!!!
   write(max)
end
```

```
> icont max.icn
> max
10
20
5
30
30
```

String Positions

- Positions within a string are between characters.
- The first position is `1`, and is to the left of the first character.

$$\begin{array}{ccccc} & h & & i & \\ \uparrow & & \uparrow & & \uparrow \\ 1 & & 2 & & 3 \end{array}$$

- `0` is also the last position of the string, and you can index from the right using negative numbers:

$$\begin{array}{ccccc} & h & & i & \\ \uparrow & & \uparrow & & \uparrow \\ -2 & & -1 & & 0 \end{array}$$

Examples

```
][ "hi"[1];
   r24 := "h"
][ "hi"[2];
   r25 := "i"
][ "hi"[3];
Failure
][ "hi"[0];
Failure
][ "hi"[-1];
   r28 := "i"
][ "hi"[-2];
   r29 := "h"
```

Substrings

- We can extract a substring from position `i` up to but not including position `j` in `s` using `s[i:j]`.
- The same syntax can be used to replace a substring with a new string: `s[i:j] := t`.
- `s[i:i] := t` inserts before position `i`.
- The range specification `i+:j` specifies a substring at position `i` of length `j`.

Examples

```
][ s := "hello";
][ s[1:3];
   r31 := "he"
][ s[1:3] := "toc";
][ s;
   r33 := "tocllo"  (string)
][ s[2] := "***";
][ s;
   r35 := "t***cllo"  (string)
][ s[1:1] := "+++";
][ s;
   r37 := "+++t***cllo"  (string)
 s[1+:5];
   r38 := "+++t*"
```

Readings and References

- Read `Christopher, pp 21--28`.