ICON

Ben Taylor
Computer Science
University of Arizona
Tucson, AZ 85719
bentaylor012@email.arizona.edu

Lize Chen
Computer Science
University of Arizona
Tucson, AZ 85721
lizechen@email.arizona.edu

April 22, 2020

ABSTRACT

This language is as cool as us!

1 Introduction

2 History

Icon is a high-level, general-purpose programming language with novel features including string scanning and goal-directed evaluation. It was chiefly designed by Ralph Griswold in 1977 at the University of Arizona. The design philosophy of Icon is to provide a “critical mass” of types and operations, free the programmer from worrying about details and put the burden of efficiency on the language implementation[1]. With Icon, the designer said he could write programs he didn’t have the time to write in C or C++.

There is no official Icon users’ group, but The Icon Project maintains a moderated "Icon-group" electronic mailing list.

3 Control Structures

Icon has a few control structures that are quite useful. They can be broken up into a couple categories, loops and conditional structures. Loops in icon are not to far gone from the loops that we are used to in java and c. The one most people are familiar with from other languages is the while loop. Like other languages it is formatted as "while (condition) do ...". The other loop that Icon uses is the until loop. The formatting is the same but instead of looping while the condition is still met, until loops until it reaches the condition statement. This means that technically "while not (condition) do ..." is the same as "until (condition) do ...".

The other type of control structures are if/then/else statements as well as switch cases. The format of the if/then/else goes...

if (condition) then (something) else (something)

This isn’t too different from what we are used to seeing in other languages. The "then" signifies what will happen if the condition is met for the if statement.

Switches are useful for thing that may need multiple if statements. An example of this is if there are multiple values that x could be and each one does something else.

```
case x of
{
    0 :write("zero")
    1 :write("zero")
    2 :write("zero")
}
```

4 Data Types

There are 12 data types in Icon.

null(n)	string(s)	co-expression(C)	table(T)
integer(i)	cset(c)	procedure(p)	set(S)
real(r)	file(f)	list(L)	record types(R)

4.1 Null

x := &null => assigns null value to x

Commonly, using &null is used to check if the node in linked list is empty.

4.2 Integer

It can enable basic operations like addition, subtraction, multiplication, division, remaindering, exponentiation, etc.

Numerical comparison operations are

N1 < N2	less than
N1 <= N2	less than or equal to
N1 = N2	equal to
N1 >= N2	greater than or equal to
N1 > N2	greater than
N1 ~= N2	not equal to

4.3 Real

All integers are real numbers, so we can use real() to convert integer to real.

4.4 String

Though Icon has syntax similar to C, it is much easier to create a string. It avoids dynamic memory allocation, so we can assign a value to it directly, e.g. x := "Hello world"

There are various functions for string, which are similar to the usage in other languages. However, Icon has an unique feature: string-scanning.

The form of a string-scanning expression is

```
expr1 ? expr2
```

Since there is no character type in Icon, string-scanning does great job when checking each character in a sting.

For example,

```
x ? {
    write(tab(0))
    while move(1) do
    write(tab(1))
}
```

This snippet is to loop each character in x, and print one by one.

In addition, using "||" for string concatenation.

4.5 Cset

Cset is like set in Java. It can remove duplicate values and sort them in an increasing order, e.g.

```
x := cset("Hello world")
write(x)
```

The output is " Hdelorw", which is based on the Ascii value of each character.

There are some simple expressions related to set theory:

A −− B	in A but not in B
B −− A	in B but not in A
A ++ B	the union of A and B
A ** B	the intersection of A and B

4.6 File

There are relatively few functions for this type. The commonest one is open(s1, s2) where s1 is the filename and s2 is the option(s).

The options[1] are:

"r"	open for reading
"w"	open for writing
"a"	open for writing in append mode
"b"	open for reading and writing
"c"	create
"t"	translate line termination sequences to linefeeds
"u"	do not translate line termination sequences to linefeeds
"p"	pipe to/from a command – UNIX

Generally, we use "r", "w", "c" more.

4.7 Co-expression

A co-expression is a data object that contains a reference to an expression and an environment for the evaluation of that expression[1].

We create a co-expression via

```
create expr
```

Take an example,

```
color := create("red"|"green"|"blue")
first := @color
second := @color
color :=^color
third := @color
```

Control is transferred to a co-expression by activating it with the operation @color [1].

Thus, the value of first is "red" and the value of second is "green". However, the value of third is not "blue" since this co-expression is reset with the operation "^". Thus, its value is "red", which means control starts from the first one again.

4.8 Procedure

Functions, which are simply built-in procedures, have type procedure[1].

In addition, we can create procedures dynamically by using proc(), e.g.

```
fun1 := proc("*", 2)
fun2 := proc("write", 1)
fun2(fun1(2, 3))
```

The output is 6.

In function proc(s1, s2), s1 represents the functionality and s2 represents the number of arguments.

4.9 List

Using "|||" for list concatenation.

This type is similar to list in other programming languages.

It has many related functions, such as get(), pop(), push(), put(), sort(), and etc.

4.10 Table

This type is similar to dictionary in Python or HashMap in Java.

It is initialized by

```
example := table()
example[key] := value
```

4.11 Set

This type is not as useful as others. It is mutable.

4.12 Record Types

A record declaration adds a type to the built-in repertoire of Icon[1].

This type is fantastic and it has to be defined outside procedures. It is very similar to struct in C, but there is no need to allocate memory when using it, which means it is much easier to use linked list in Icon.

Take an example,

```
record student(name, age, gender, id)

procedure main()
    one := student("Alice", 18 ,"F" ,123)
    write(one.name)
    write(one.id)
end
```

We can access the value stored in record by calling record_name.value_name.

4.13 Type Conversion

Csets, integers, real numbers, and strings can be converted to values of other types[1].

The possible type conversions are given in the following table.

type in	*type out*			
	cset	integer	real	string
cset	=	?	?	√
integer	√	=	?	√
real	√	√	=	√
string	√	?	?	=

The symbol ? indicates a conversion that may or may not be possible, depending on the value.

5 Subprograms

6 Summary

References

[1] Ralph E. Griswold and Madge T. Griswold. *The ICON Programming Language*. Annabooks, 3rd edition, 1996.

[2] Laurence Tratt. Experiences with an icon-like expression evaluation system. In William D. Clinger, editor, *Proceedings of the 6th Symposium on Dynamic Languages, DLS 2010, October 18, 2010, Reno, Nevada, USA*, pages 73–80. ACM, 2010.