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 30, 2020

ABSTRACT

The goal of this project was to gain all the information that would be relevant to programming in the the high level language of ICON. Throughout we explain the things that we learned including how functions (procedures) work, as well as the typing of variables, control structures, and all the other odds and ends that makes the ICON language unique. It can be seen that ICON is built to be simple to use, and allow for fast and rather painless programming sessions.

1 Introduction

Icon in a very-high-level programming language that was developed in 1977 by Ralph Griswold at our home school, The University of Arizona. Icon is very similar structurally to C as well as having some features that resemble Python. Icon offers many interesting features such as type-less variables and as well as a lot of useful sting operations that you do not get in many other languages at the time and even a little bit right now. These make Icon a great language to pick up in order to quickly throw together a program idea without having to spend the time needed with a lower level language.

2 History

Icon 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[3]. With Icon, the designer said he could write programs he didn’t have the time to write in C or C++. While there is no official Icon users’ group, The Icon Project maintains a moderated "Icon-group" electronic mailing list. And even though it was initially developed around 43 years ago, it is still being worked on with the last stable release version being put out on September 27, 2018.

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("one")
    2 :write("two")
}
```

4 Data Types

There are 12 data types in Icon, these include,

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)

Variable types such as int, real, and string, etc. do not have to be declared. Instead the language interprets the type at each operation. If an item can do the operation it will continue the program if not it will throw an error or exception. In other words x := 7 can be interpreted as a string, real, or integer based on the operation.

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).

We only list the frequently-used options:

"r"	open for reading
"w"	open for writing
"c"	create

Generally, we use "r", "w", "c" more, and we can combine them into one statement: open(filename, "rwc").

There is a clear code example in p3_file.ico which contains almost all the related functions.

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[3].

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 [3].

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[3].

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[3].

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. The possible type conversions are given in the following table[3].

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

The symbol ? means the conversion is based on the value. For instance a 7 can be converted from a string to a number or a number to a string, however "7a" cannot be converted from a string to a number unless you take the substring of the first element.

5 Subprograms

A subprogram is just like a main program, which can be compiled independent of the main program[4].

In Icon, there are two kinds of subprograms, procedures and records.

5.1 Procedures

Procedure is everywhere in Icon programming. Both main and helper functions start with "procedure". We can consider it as definition. After that, the defined functions can be invoked.

For example,

```
procedure main()
    helper()
end

procedure helper()
    write("Hello world")
end
```

In this example, helper is a subprogram of main function.

5.2 Records

A record is like class in Java or a struct in C. It is mainly used to store data, in such a way that the user can then access or modify it easily. One of the advantages of records is to avoid dynamic memory allocation. p4_count.ico is an example which reads integers and sorts them in a linked list.

5.3 Include

The term "include" in Icon is similar to the one in C programming. We can use it to link two files together to form one executable run-unit.

For example,

program.ico

```
$include "help.ico"

procedure main()
    local temp
    help()
    temp := Student("Ben", 19)
    write(temp.name, " ", temp.age)
end
```

helper.ico

```
record Student(name,age)

procedure help()
    write("nothing")
end
```

In this example, we can access the functions in helper.ico when running program.ico. Thus, using "include" can improve readability.

6 Summary

Icon is a high-level, general-purpose programming language, developed in Arizona, with novel features including string scanning and manipulation as well as goal-directed evaluation. Simply put, Icon was designed to make working with string inputs quick and convenient. It can do this by having many predefined functions and operations for working with strings, these include a simple input (read()), output (write()), and concatenation (||). The other key feature of Icon is the goal-directed evaluation. This basically stating that for statements that may include a Boolean output it does not evaluate each individual process as a Boolean, but rather by weather it succeeds or fails. If it succeeds all the way through then it will return a value equal to true, and if it fails then it will return a value equivalent to false. String scanning was one of the other key features of Icon. It follows the expression (expr1 ? expr2) and since there are no data types to worry about, string scanning doesn't have to get hitched up when looking through the elements of a string. Along with making string manipulation as well as scanning very simple, the entire programming languages style is very simple to learn and quick and and intuitive to program with.

References

[1] Thomas W. Christopher. *Icon Programming Language Handbook*. Dr. Thomas W. Christopher, Tools of Computing LLC, 1996.

[2] Ralph E Griswold and Madge T Griswold. *The Implementation of the Icon Programming Language*. Princeton University Press, USA, 1986.

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

[4] Subrata Ray. *Fortran 2018 with Parallel Programming*. Internet resource, 2020.

[5] 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.