ICON PROGRAMMING LANGUAGE

Madison Ridley
madisonridley@email.arizona.edu

Sam Bryant
sbryant1@email.arizona.edu

April 30, 2020

ABSTRACT

This paper is a general overview of the Icon programming language. Icon is a programming language developed specifically to handle symbolic data and expressing it both textually and graphically. It serves as a general purpose programming language that provides quick solutions to programming problems. This paper will delve into the history of the language and its various control structures, data types, and subprograms which mark is as unique from other languages. The sources used to inform this paper and the information is provides are listed as references at the end.

1 Introduction

Icon is a programming language for general use that is primarily focused on handling data. It focuses on representation this data in either a textual or graphical format. The syntax of Icon is very similar to C though it differs from C greatly in regard to the features and functions of the programming language. There are many features that are unique to Icon and together form a language that delves into different possibilities and uses than others before and after it.

In order to fully appreciate the content in this paper, a reader should have some type of knowledge about programming languages and some idea of the structure and workings of at least one language. There are many areas of Icon that show similarity to Python and other areas which emulate some of the functionality of C.

2 History

Icon was developed by Ralph Griswold and David Hanson with the first version of the language released in 1977. Since its initial release, there have been a total of nine versions. According to Griswold, he wanted to create a language that contained entirely unique features of its own which would mean it had additions singular to it. He chose this as opposed to simply emulating other more established languages with often used features. There are of course many structures and implementations that are quite similar to other languages, but Icon also contains many nuances that make it stand out as quite different from others. SNOBOL and SNOBOL5 have been attributed as large influences on the development of Icon's functionality. Icon seeks to create more general implementations of some of SNOBOL's already established mechanisms. There are many aspects of this language, though Griswold has said one of the main functions of the language is to deal with general tasks involving strings and data structures. This, along with other features will be explored throughout the remaining sections of the paper.

Currently, the ninth version of Icon is run on many operating systems. There is also now a language, Jcon which is a Java-based implementation of the Icon language, proving that while it is not as widely used as some languages, there are still current uses for its unique features.

3 Control Structures

Icon uses expressions when evaluating for control structures, which makes it different from many other programming languages. As opposed to using Boolean's to drive control structures, Icon uses the success or failure of

an expression to do so. It evaluates expressions for control structures using backtracking, and it evaluates in a way described as “cross-product evaluation with depth-first search”. What this means is that each expression is evaluated from left to right, ensuring each is successful before moving on, and then any comparisons are used in the evaluation. At each step the success of the overall expression is checked before moving on. This provides some interesting options because it allows for much freedom involving what expression should be evaluated. It also means that control structures have to be considered in a different manner than they are in other languages since Boolean’s are no longer a part of their functionality. Braces can be used in Icon programs, however they are not a necessary component for the code to run. In control structures, braces can be used to contain multiple expressions.

There are two main categories of control structures in Icon which include more specific structures that are described in more detail below.

3.1 Loops

Included in Icon’s control structures, there are several types of loops which evaluate expressions multiple times. Two of these are "while" and "until". Both have a similar structure, as shown in the example code below.

EXAMPLE:

```
procedure main()
    i := 0
    while i < 10 do
        write(i)
        i := i + 1
end
```

```
procedure main()
    i := 0
    until (i == 10) do
        write(i)
        i +:= 1
end
```

While both examples include the keyword "do", these structures don’t require its use; it is optional to include. "While" will execute as long as the included expression is successful, and "until" will halt execution once the expression is successful. This is the main, notable difference between the two options. There are several additional control structures which are often used in tandem with the loops. One example is "not". This will succeed if the inside expression fails. Another helpful addition is "next". This can be used in an instance where a program may want to skip the current value when looping. Using the "every" control structure provides another way for programs to execute code multiple times. A variable will iterate until it reaches the end condition, in which case the execution of the loop will stop.

EXAMPLE:

```
procedure main()
    every i := 1 to 10 do
        write(i)
end
```

The last control structure that is used for loops is "repeat". This is the only method which continues to execute with no regard to whether the expression is successful or not. The only way to stop the repetition is with the use of "break".

EXAMPLE:

```
procedure main()
    i := 0
    repeat {
        write(i)
        if (i > 10) then break
        else i +:= 2
    }
end
```

3.2 Selections

Other than loops, there are several other control structures that can be used throughout icon programs. A key structure is "if-then-else". Much like other languages, this control structure is used to conditionally execute expressions. The "if" statement is evaluated, and should the expression be successful, the program moves on to execute the code included in the "then" statement that follows. Otherwise, the code included with the "else" statement will be executed. It is not a requirement to include "else", so in those cases, no code will execute should the initial expression fail.

EXAMPLE:

```
procedure main()
    i := 1
    write("i is 1")
    if i < 0 then write("i is less than 0")
    else write("i is greater than 0")
end
```

Lastly, Icon can use "case" to execute selection based on a specified value. It includes an expression which, when evaluated, is compared to the various possibilities provided.

EXAMPLE:

```
procedure main()
    x := 117
    case x of {
        8 : write("x is 8")
        28 : write("x is 28")
        42 : write("x is 42")
        117 : write("x is 117")
        default : write("No number found")
    }
end
```

4 Data Types

In total, Icon contains twelve different varying data types. Many of these are types seen in many other languages or are mostly equivalent to something seen before with there being a lot of similarities seen between Icon and Python in particular. However, there are a few unique types and modifications that are unique to Icon as well. There also remains the option to define program specific data types within the code. Additionally, there are also several helpful functions that can be used in regard to data types. Icon can determine the type of a value by using the built in function type(x), and the keyword 'record' is used to add a user-defined data type to a specific program.

4.1 Co-Expression

This is a data type that is unique to the Icon programming language. It uses 'create' in order to assign values to expressions that can be used at any time. This data type is specifically the part of the program that is running independently from the others. It can be used as a co-routine or a generater. It provides a unique way to assign values and run programs that is quite different than other types. The example code below shows a way that Co-Expressions can be used within a procedure.

EXAMPLE:

```
procedure co_expression(num1,num2)
    nbr1 := create add1(num1)
    nbr2 := create add2(num2)
end
```

4.2 Set

Sets are a structure which can hold elements of any type. They are not organized in a particular order, and since they are stored in no definitive order, they cannot be accessed by a numerical index. Their defining factor is that they cannot contain multiple equivalent elements. They way we can create a set is by using the syntax set(List) which returns the list included as a parameter to a set containing its unique elements.

4.3 Cset

Csets are Character Sets. They cannot contain multiples just like a regularly defined set in Icon. They also have no definitive order. Essentially, they only defining difference between a cset and a set is that a cset can only contain characters. It is also limited to characters that are 8 bits. There are several advantages to this data type. Characters like newline and tab can be included in them creating a easy way to keep track of such character values.

4.4 List

This type is similar to its counterpart included in most other languages. In Icon, lists can contain any multitude of types. In a single list, there can be multiple types; it is not necessary for them to match. This is very similar to the Python programming language in the freedom that it allows as far as how a programmer can populate a list.

4.5 Null

This is a singular value defined by Icon. Often times it is used as a default value if the user does not specify a function return value, or there is an argument that has not been specified. It cannot be used in any regular expressions as doing so will cause an error. Null is a data type or keyword that exists in most other programming languages, though it is sometimes referenced by another keyword such as 'None' in Python.

4.6 File

Files are used to read in external files to the program, and file specific operations can be performed using this data type. They can be read or written to, and their use in Icon is similar to other languages that also contain the ability to open and edit files.

4.7 Procedure

A procedure is a data type which specifically represents a value. This value can be assigned to a variable.

4.8 Integer

Integers in Icon are stored in 32-bits, and the arithmetic operators used are similar to other programming languages. The bit-wise operators differ slightly, including syntax such as iand(a,b) which returns the result of bit-wise 'and' on the parameters a and b. The rest of the operations follow a similar syntactical pattern.

4.9 Real

Real is essentially the same as Icon integers with the only difference being that their size is dependent on the computer the program is being run on. This is very similar to how C defines different integer values.

4.10 String

Strings are defined in Icon by using the double quotation marks. Unlike in many other languages, strings are a data type on their own rather than simply a list of characters. There are quite a few functions applicable to strings, and one of Icon's main selling points is the versatility with which a programmer can manipulate strings.

4.11 Table

A table is a type which maps a key to a corresponding value. It is similar to a map or a dictionary in other languages and provides a very similar functionality. It can be created by using table(n). The parameter n will be used as the default value for any elements that are added to the table without a specified value of their own. This is especially helpful when a programmer may want to initialize the values of the keys in their table to the same value.

4.12 Window

Windows are included as a part of Icon graphics. They can be written to, drawn on, and otherwise display graphics. Icon has a few built in graphics functions, but most can be used when including the graphics library before opening a window. Windows are easily opened using the specified dimensions, and then can be used to design a multitude of graphics. Icon makes a point of including relatively simple and easy to include graphics to appeal to programmers.

5 Subprograms

Icon has several subprograms included in the format of the language. Mainly seen are procedures, which are used frequently throughout full programs in Icon. These procedures are used to perform an action, and can be assigned as a value to a variable. Procedures are statically compiled separately from the rest of the program. Icon also includes the use of a multitude of built-in functions. These include mathematical operations like abs() which computes the absolute value of a numeric value. It also includes many instances of functions that can be used for string manipulation, something that Icon focusing on providing with much ease of use. There are a few functions which use backtracking in order to run, but the vast majority of them do not. Functions and procedures both have the option of including parameters. These parameters can either be passed by value if they are a primitive type or by reference if they are a structure. Icon does not include the use of classes, thus it can not be used for Object Oriented Programming.

6 Summary

Overall, Icon is a programming language that was developed with the intention to be unique. It employs many features and functions that other languages do not heavily rely on. Its built in backtracking and focus on handling symbolic data makes it stand out among other languages which simply repeat what has already been done. This general purpose language certainly meets its goals to create a simplistic way to handle data in a way that promotes ease of use and quick solutions to programs.

References

[1] Bob Alexander. Icon programming language reference. `https://www2.cs.arizona.edu/icon/refernce/ref.htm`.

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

[3] Todd A. Proebsting Gregg M. Townsend. Jcon: A java-based implementation of icon, 1999. `http://www.cs.arizona.edu/icon/docs/ipd286.htm`.

[4] Mariya Mykhailova. Programming language icon, 2012. `http://progopedia.com/language/icon/`.

[5] NA. The icon programming language. `https://www2.cs.arizona.edu/icon/`.

[6] Madge T. Griswold Ralph E. Griswold. *The Icon Programming Language*. Peer-to-Peer Communications, 3 edition, 2002.

[7] Carl Sturtivant. Experimental native distribution of icon for microsoft windows, 2015. `https://www2.cs.arizona.edu/icon/v95w.htm`.

[1] [2] [3] [5] [4] [6] [7]