1 INTERCAL

Intercal [Woods 73], a language invented by Donald Woods and James Lyon, is the short name for "Compiler language with no pronounceable acronym". Its goal is to have nothing at all in common with any other programming language. As such, it presents a refreshing view of what a programming language might be, but, luckily, usually is not. Intercal has novel syntax as well as semantics. It is intended to be as obscure as possible while still retaining some modicum of function.

Intercal chooses to give unusual names to the characters available for writing programs. For example, the comma is called "tail", and the semicolon is called "hybrid". I will avoid these names for clarity; the manual has no such compunctions.

Although Intercal was intended as a joke, some actual languages share some of Intercal's misfeatures. I point these out below.

1.1 Data structures

There are two basic types, unsigned 16-bit integer and unsigned 32-bit integer.[1] Integer literals are formed with the '#' modifier followed by digits. The type of the literal is deduced from the value represented by the digits.

Figure 0.0

#34	1
#01	2
#12345678	3

Variables are not declared; the type of a variable is determined by its use. The identifier for a variable is a modifier followed by an integer in the range 1 to 65535. Modifiers for unstructured variables are the period ('.') for 16-bit integers and the colon (':') for 32-bit integers. The variables .34 and :34 are distinct (they have different types and may coexist in the same program). The variables .45 and .045 are identical.

Modifiers for array variables are the comma (',') for 16-bit integer arrays and the semicolon (';') for 32-bit integer arrays. Subscripts are prefixed by the word sub, as in ,34 sub #23 #42 (equivalent to array34[23,42] in more conventional languages). Arrays are not declared or preallocated; they must be allocated at runtime.

---

[1] There are languages such as Bliss with only one data type, integer.

1.2 Operators

There are five logical operators, carefully designed to make arithmetic nearly, but not completely, impossible.

The interleave operator is represented by '¢', although the standard compiler accepts '$' as well. It takes two 16-bit operands and produces the 32-bit quantity formed by alternating the bits of the operands. The most-significant bit of the result is the most-significant bit of the first operand.

The select operator is represented by '~'. It takes either 16-bit or 32-bit operands, which it coerces to 32 bits. Those bits in the first operand corresponding to 1's in the second operand are selected and packed in the least-significant bits of the result, which is either 16 bits or 32 bits, depending on the number of bits selected. For example, #201 ~ #179 selects from 11001001 the bits specified by 10110011, namely, 10001, or #17.

The other three operators are unary. Each performs its operation on pairs of adjacent bits, with the result appearing in the position of the lower-order bit of the two. Another way of describing the action is to take the 16-bit or 32-bit operand, circularly right shifting it one bit, then performing the operation on the original operand and its shifted image. The three operations are and (represented by '&'), or (represented by 'V'), and xor (represented by 'V'). Unary operators are **enclitic**, that is, they are placed between the modifier and the integer that together make up a constant or a variable. For example, #V123 (binary 1111011) has the value 32895 (binary 1000000001111111). It is invalid to place more than one enclitic after the modifier.

There are no rules of precedence; all potentially ambiguous expressions must be parenthesized. There are two types of parentheses: '"' and '".[2] They must be alternated when nested so the parser can always determine whether a particular occurrence of " or ' is an opening or a closing instance. For example,

Figure 0.0

```
,1 sub ",2 sub ',3 sub #1'" #22                                    1
```

would be written in a more conventional language as

Figure 0.0

```
array1[array2[array3[1]], 22]                                      1
```

Unary operators applied to parenthesized expressions are also enclitic

[2] The shell programming languages in Unix (sh and its descendents, including Perl) also use these two types of parentheses, but they do not nest and have slightly different semantics. Here is one case where Intercal is *less* obscure than an actual language.

and are placed after the opening ' or ". The combination ".' may be rewritten as ! for conciseness (and obscurity, of course).

1.3 Statements

Statements are free format.[3] Spaces and newlines in input are optional and ignored, even within literals and identifiers.[4] The format of a statement (optional parts are shown in brackets) is

Figure 0.0

```
[(label)] identifier [fudge] body                                    1
```

The label is a number in the range 1..65535, although it is wise to avoid numbers between 1000 and 1999, which are used by the library.[5] The identifier indicates to the compiler that a statement is starting. The valid identifiers are do, please, and please do. The compiler complains if too low or too high a fraction of the statements use a polite form. The fudge is either a percent chance of execution (like %76) or a request not to execute the statement at all, either not or n't. Such a statement is abstained from as execution begins, but abstinence can be overridden, as you will see.

Assignment is performed by an arrow ('<-').[6] 32-bit quantities are coerced to 16 bits only if the value is less than 65536. Arrays are allocated by assignment to the array variable; in this case, the right-hand side indicates the dimensions, not the contents. The following example builds two 32-bit arrays; one has 10 elements, the other 100.

Figure 0.0

```
do ;1 <- #10                                                         1
do ;2 <- #10 by #10                                                  2
```

Control transfer is performed by the next statement:

---

[3] Before we say that Intercal is superior in this regard to FORTRAN, note that Miranda, certainly a modern language, and Mumps (not so modern), are not only statement-per-line languages, but also use indentation for grouping. See the discussion of grouping by indentation later in this chapter.

[4] FORTRAN shares this misfeature. Consider the FORTRAN statement do 100 j = 1.10. The fact that 1 and 10 are separated by a period instead of a comma makes this an assignment statement equivalent to do100j = 1.10.

[5] The IBM 1620 computer reserved some locations in low memory for arithmetic tables; it was possible, although very unwise, to overwrite these.

[6] In this regard, Intercal is more advanced than C, in which programmers commonly interchange '=' and '==' by accident. The problem is severe in C because when either of these symbols is permitted, so is the other one, often with a disastrously different meaning.

Figure 0.0

```
        do (49) next                                    1
```

This statement has two effects. First, the location of the following statement is pushed on a stack. Second, control is transferred to the indicated label (in this example, 49). The stack has capacity for 79 entries; the next statement fails if the stack is full.[7] The error message is "Program has disappeared into the black lagoon".[8]

Any number of elements can be popped from the stack by the forget statement:

Figure 0.0

```
        do forget #3                                    1
```

This example pops 3 elements. It is not erroneous to pop more values than the stack contains; the effect is to clear the stack. Unconditional jumps are generally followed by a forget #1 statement to avoid cluttering the limited stack.

Procedure return is performed by the resume statement, which is like forget except that after popping, it jumps to the statement addressed by the last entry popped. If more entries need to be popped than are on the stack, the program terminates. The effect of a computed goto can be programmed by the following code:

Figure 0.0

```
        do note .1 has value 1..4, the jump code        1
        do (3) next                                     2
        do note code for case .1 = #4 goes here         3
        do (101) next                                   4
(3)     do (2) next please forget #1                    5
        please note code for case .1 = #3 goes here     6
        do (101) next                                   7
(2)     do (1) next please forget #2                    8
        please note code for case .1 = #2 goes here     9
        do (101) next                                   10
```

---

[7] This intrusion of implementation limits on the language itself is one end of a spectrum. At the other, more familiar end, programs that use lots of stack resources might or might not run depending on the implementation; there is no way the programmer can tell. Ada sits somewhere in the middle, providing runtime library routines that report some limits (particularly numeric ranges), but not all (such as available stack space).

[8] Unhelpful error messages are legion. For example, a common compiler diagnostic in SAIL is "The details of this error can be found in your code". Many older compilers report only error codes, not messages, and the user must look the codes up in a manual.

```
(1)         do (100) next please forget #3                    11
            please note code for case .1 = #1 goes here       12
            do (101) next                                     13
(100)   do resume .1                                          14
(101)   do forget #1                                          15
```

Line 1 is a comment, since it is abstained from (starting with do not), so the unrecognizable remainder of the statement is never executed and never causes problems.

Variables may be saved and restored, which is helpful for passing parameters to procedures. The stash statement takes a list of variables (separated by '+') to be stacked; the retrieve statement takes a list of variables and restores the most recent value stashed for each. A variable may be stashed more than once; there is an implicit stack for each variable, and retrieve only pops one value. It is invalid to pop more values than were pushed for any variable. The error message is "Throw stick before retrieving".

Variables can be temporarily deactivated by the ignore statement, causing assignments and input to leave those variables unchanged, although they may be accessed. The variables can be reactivated by the remember statement.

Intercal does not have conditional statements, so it is difficult to skip around code. The ignore statement helps, but the abstain statement is often needed. It can effect abstinence on a particular statement (by specifying its label) or a class of statements (by specifying the keywords):

Figure 0.0

```
please abstain from stashing                                  1
do abstain from (34)                                          2
please do abstain from nexting                                3
please abstain from abstaining                                4
please abstain from reinstating                               5
```

Line 4 is valid, but surprising. Line 5 is valid, but most likely a bad idea. Abstinence is terminated by a reinstate statement. It is possible to reinstate a statement that is abstained from because it has a not in it. It is not necessary to reinstate by the same mechanism that started the abstinence; in the previous example, statement 34, abstained from in line 2, could be reinstated by do reinstate nexting.

A program is usually terminated by a give up statement, equivalent to resume #80. This statement cannot be abstained from by keyword, nor can it be reinstated at all. So don't give up is always a do-nothing statement.

Input and output are performed by the write in and readout statements, respectively. Each input number must be on its own line. The input format is English spelling; that is, #3402 is input as three four oh

(or zero) two. The output format is extended Roman numerals.[9] An overline indicates multiplication by 1,000; lower case indicates times 1,000,000; zero is an overline itself; I and M both mean 1,000, but M may only be used for numbers less than 4,000 mod 1,000,000.

1.4 Final comments

Intercal is a very funny language and is also quite a challenge for the programmer. The manual includes a listing of the subroutine library, used for such operations as addition, subtraction, multiplication, and generating random numbers (this latter using a statement with fudge %50). It is absolutely cryptic, but, amazingly, shows that arithmetic is possible. Intercal has been extended to deal with different base number systems; for example, Intercal-7 specializes in base-7 representation.

[Woods 73] DONALD R. WOODS AND JAMES M. LYON, *The Intercal Programming Language Reference Manual* (Not published) (1973).

[9] Languages often have difficulty creating readable output. For example, enumeration types are frequently coerced to integers, which leads to obscure output.