On The Preservation Of The Arithmetic IF

Arnold D. Robbins

Friends Of The Arithmetic IF
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

UUCP: { akgua, allegra, hplabs, ihnp4 }!gatech!arnold
ARPA: arnold%gatech.csnet@csnet-relay.arpa
CSNET: arnold@gatech

ABSTRACT

The Arithmetic IF was the earliest "high-level" language statement for altering flow of control in a program. We argue that this construct should not only be preserved, but enhanced, and incorporated into newer languages.

January 7, 1985

On The Preservation Of The Arithmetic IF

Arnold D. Robbins

Friends Of The Arithmetic IF
School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

UUCP: { akgua, allegra, hplabs, ihnp4 }!gatech!arnold
ARPA: arnold%gatech.csnet@csnet-relay.arpa
CSNET: arnold@gatech

1. INTRODUCTION

Many younger computer scientists, indoctrinated to programming the easy way, with Pascal, C, or some other wimpy language like Algol 68, often learn FORTRAN as well. However, a feature of FORTRAN that is frequently debased, and almost always never taught, is the much maligned Arithmetic IF. For those who have never seen it, it works as follows:

```
      INTEGER J
            ...
      J = some value
      IF (J) 10, 20, 30
10    Something
      ...
20    Something Else
      ...
30    Yet Another Thing
      ...
```

If J is negative, control transfers to line 10. If J is zero, control transfers to line 20. If J is positive, control transfers to line 30. What could be clearer than this simple construct?

This paper presents an argument for preserving the Arithmetic IF, as well as possible formulations for adding it to more "modern" programming languages.

2. A SENSE OF HISTORY

Computing Science is one of the fastest changing disciplines in the world today. More has been accomplished in the past 30 years with computers than in the past 500 with Physics or Mathematics.

This is indeed something to be proud of, yet, we computer scientists should be careful to preserve a sense of history as we design the next generation of computer languages and software systems. Should we throw away a useful feature, just because it is old? Have we no sense of tradition, of history? We maintain that good Computing Science can be accomplished through a harmonious blending of the old and the new.

3. EFFICIENCY

Amid the never-ending quest for elusive things like "structure" and "elegance", we must also not forget the efficiency of our programs. The Arithmetic IF will compile into about four instructions on most machines, possibly even less on others. As a generic example:

```
LOAD      R1,J   load J into register
BLT  L10  branch to 10 if negative
BZ   L20  branch to 20 if zero
BR   L30  unconditional branch to 30
```

On certain machines, the Arithmetic IF compiles into even better code. For instance on Prime computers, one may use the V-mode CAS (Compare Accumulator and Skip) instruction. This instruction takes four input lines; the CAS opcode and three labels to branch to, if the Accumulator is negative zero or positive, respectively. All in the microcode!

4. OTHER LANGUAGES

We feel that many contemporary languages would benefit from the Arithmetic IF. The following discussion presents formulations for an equivalent construct to be incorporated into some of the popular modern-day languages.

4.1. Pascal

Pascal is one of the more widely used languages in Universities and in Industry today. The Arithmetic IF would even look the same in Pascal as it does in FORTRAN (with the exception of label declarations and the semi-colon at the end). Surely homogeniety of language constructs across programming languages is a desirable feature! Since ISO is currently considering extensions to standard Pascal, we suggest the following:

```
{ FORTRAN doesn't require this part }
label
        10, 20, 30;

if (<expression>) 10, 20, 30 ;
```

A missing label would be a compile time error, as would an *else* part, while the use of any expression that is not integral (e.g. a *real* variable in the *if*) would be "undefined."

4.2. C

The Arithmetic IF would fit in with C fairly well, although not nearly as nicely as it does in Pascal.

```
if (<expression>)
        label1:, label2:, label3:;
```

In C, if any label is missing, the default action for the corresponding value would be to fall through to the next statement (analogous to missing expressions inside the C *for* loop). Thus the Arithmetic IF in C could have a corresponding *else* part. This part would be executed if a label was missing and the expression in the *if* corresponded to that label. If no label is missing, the *else* would still be syntactically valid, however the code inside would never be reached (unless one of the labels is there). A smart compiler could warn about this. The labels are followed by colons so that they can be syntactically distinguished as labels, instead of looking like expressions (the rvalues of the named identifiers, used with the comma operator).

4.3. Ada (tm)*

Finally, in keeping with the advanced nature of the Ada (tm) programming language, we present an enhanced version of the Arithmetic IF to be incorporated into that language. We term it the "Enumerated IF." The Enumerated IF could be used on any enumerated type.

---

* "Ada" is a trademark of The Ada Joint Program Office of The United States Department of Defense. So there.

```
type fruit is (apple, orange, pear, mango, melon);
food : fruit;  -- declare a variable

If (food) Then
        label1, label2, ... , labelN;
Endif;
```

This would work like you might expect; if *food* is an *apple*, control goes to *label1*, and so on. By default, the labels correspond to the enumerated values, in order. There must be a label for each possible enumerated value. This can be changed; the Enumerated IF also has a variant allowing explicit correspondence of labels to enumerated values.

```
type fruit is (apple, orange, pear, mango, melon);
food : fruit;  -- declare a variable

If (food) Then
        label1,
        orange .. pear => label2,
        melon => label4,
        mango => label5;
Endif;
```

This example shows that you can have default correspondences, a range applied to a label, and explicit correspondences. Once a named correspondence has been given (value => label), all the rest must also be named. This is keeping with the rules for passing parameters to procedures and functions. If some values do not have corresponding labels, and the expression inside the *if* takes one of these values, the associated *else* clause would be executed. If there is no *else* clause, the exception 'MISSING_ENUMERATION_LABEL' would be raised.

While some might argue that this functionality is already available via the Ada (tm) *case* statement, we feel that the Enumerated IF is clearer. In the Enumerated IF, the control flow is separated from the actions to be done. In the *case*, control flow is mixed with action statements, i.e. code to be executed at each point. We feel that the Enumerated IF provides a cleaner separation between the two problems of "where to go", and "what to do when you get there."

A somewhat less extended version of this construct might also be added to Pascal, but it clearly is most useful in Ada (tm).

5. FORTRAN 8x

Recent rumor has it that the ANSI X3J3 FORTRAN committee is considering either removing the Arithmetic IF entirely, or flagging the construct as one that will not be allowed in the language revision after FORTRAN 8x.

This is terrible! It would be absolutely silly to add the Arithmetic IF to other languages, and then drop it from FORTRAN! If all of the reasons we stated above apply to "modern" languages, how much the more so do they apply to FORTRAN itself!?! In short, we respectfully request that X3J3 leave the venerable Arithmetic IF alone, instead of deprecating it.

6. CONCLUSION

We have presented some arguments in favor of preserving the Arithmetic IF, and have also shown how it might be easily added to the so-called more "modern" programming languages. In particular, Ada (tm) offers the greatest opportunity for increasing the flexibility and usefulness of this construct.

7. ACKNOWLEDGEMENTS

I would like to thank Doctors Richard LeBlanc and Martin McKendry for *not* reading this paper, as they would have undoubtedly thrown it into the "circular file" if they had.