STRUCTURED TRANSFER OF CONTROL

David O. Williams
Data Handling Division
CERN, CH-1211 Geneva 23

Abstract

A proposal is presented for the generalisation of certain control structures by permitting the (optional) execution of a block of code:

(i) at each new cycle in a loop
(ii) following termination of a loop
(iii) following termination of the "normal" code of a subprogram/procedure.

1. Introduction

In a related paper (Alternate RETURNs) I discuss how I started to investigate the arguments for and against the Alternate RETURN concept of FORTRAN-77. As part of that work I became interested in the reasons why statement labels and the GOTO statement refuse to die in mainline procedural languages. I concluded that the possibilities for structured transfer of control offered by these languages are too primitive, and I would like to start a discussion on some more powerful concepts.

2. Problems with Present Control Structures

2.1 GOTOs Considered Harmful

A long time has passed since Dijkstra wrote his famous letter [Dijkstra 68]. I have myself been responsible for part of the development and maintenance of a 16,000 line PASCAL program which contains no GOTO statement. Why then do the bodies which are responsible for the definition of the mainline procedural languages (ADA, FORTRAN, PASCAL, for example) still cling to GOTO statements?

Some people will, of course, answer that it is in inherent in the conservative nature of these bodies to retain outdated concepts. However I believe that there is a much more fundamental reason. The action described by the GOTO statement is a primitive action. Any existing or future control structure can be decomposed into a series of GOTO statements, conditionally guarded as appropriate. The reluctance to finally condemn the GOTO statement reflects, I believe, a gut feeling that control structures in existing languages are not sufficiently powerful, and that we need GOTO statements in order to adequately treat the problems of the real world.

2.2 Deeply Nested IF-THEN-ELSE Considered Harmful

Many people will, of course, deny the validity of this last remark, and indeed it is, formally speaking, incorrect. Any control structure of arbitrary complexity can be mapped onto the combination of a generalised LOOP structure, taken together with an appropriate number of IF-THEN-ELSE conditions. However, when several nested levels of IF-THEN-ELSE structures are required to express the problem at hand, then the code is often difficult to write and confusing to read.

There are several interesting recent papers in SIGPLAN notices [Hill 82, den Hertog et al 83, Lakhotia 83, Yuen 83, Rosenbloom 83, Newman 83, Amit 84, Yuen 84] which address this issue, or closely related points. My personal summing-up of the theme of these papers is the following: at the point where the programmer has to "invent" a boolean variable merely in order to map the problem onto the available control structures, then this "invented" variable is as dangerous and confusing as the poorly regarded GOTO statement. For a clear example of "invented" variables see STAYINLOOP and CONTROL in [Yuen 84].

2.3 More Powerful Control Structures

Many of the papers mentioned above make proposals for resolving the problem of deeply-nested IF-THEN-ELSE structures. However it is my belief that they are not radical enough. The real cause of "invented" boolean variables, it seems to me, is that our loop constructs are too primitive, especially in the sense of not allowing the programmer control of the behaviour at loop exit time.

FORTRAN-8x currently proposes to define a generalised loop structure, with three sub-forms, the indefinite loop (DO [loop_ID]...REPEAT), the counted loop (DO n TIMES...REPEAT), and the indexed loop (DO I=...REPEAT). There are related statements EXIT [loop_ID] and CYCLE [loop_ID] to exit the (designated) loop and to cause the next iteration, respectively.

The problem that I perceive with this proposal is that the programmer sometimes, but not always, wants to execute certain code at the time of loop exit or loop cycling. This behaviour is essentially structured. To achieve it in the case of EXIT, a boolean variable will have to be "invented". For the CYCLE case, things are worse, since not only must a control variable be "invented", but in addition the CYCLE syntax forces the code to be placed at the head of the loop, rather than at the tail, where it probably belongs on logical grounds.

3. Proposed Modification to Control Structures

I propose that in the following cases:

(i) next cycle of a loop
(ii) exit from a loop
(iii) return from a subprogram/procedure

the programmer should always be able to specify the (optional) execution of a block of code.

It is my conviction that this approach would render unnecessary most, and probably all, cases of the "invention" of boolean variables, and render the code simpler to understand, both for writer and reader. I would be interested to have someone more knowledgeable than myself in complexity metrics attempt to justify this statement.

I have attempted to provide a syntax for such control structure exits, in the spirit of the FORTRAN-8x proposal, in order to give readers a feeling for the usefulness, or otherwise, of this idea. Conversion to other procedural languages is not very difficult.

3.1 Identification of the Code Blocks

This is achieved through ON_structure, DONE_structure, bracketing keywords. Specifically ON_REPEAT, DONE_REPEAT for the next cycle of a loop, ON_EXIT_LOOP, DONE_EXIT_LOOP for the exit of a generalised loop, and ON_EXIT_CODE, DONE_EXIT_CODE for the return from the body of a subprogram/procedure.

As discussed below, the DONE_REPEAT statement is probably better suppressed, since it is redundant with respect to END_LOOP. I should also say that I do not feel strongly about the choice between DONE_structure and END_structure as the terminating keyword.

3.2 Optional execution of the Code Blocks

The code blocks will be entered when an associated EXIT_structure statement is executed. Specifically EXIT_LOOP will cause entry into the ON_EXIT_LOOP block, and EXIT_CODE will cause entry into the ON_EXIT_CODE block. The QUIT_structure statement (where quit implies to me a faster, less ordered departure than exit) leaves the structure without entering the ON_structure/DONE_structure code block.

I assume that in cases of "normal" structure exit, such as completion of an indexed or counted loop, execution of the ON_structure code block will usually be what the programmer wants. However for the sake of generality it should be possible to avoid this, by appending QUIT to the END_LOOP statement. Note that, in any case, the programmer is not forced to provide an ON_structure code block.

3.3 Loop Control

I would suggest LOOP, END_LOOP as the bracketing keywords, REPEAT as the action statement (in place of CYCLE in the present FORTRAN-8x proposal), ON_REPEAT to introduce the REPEAT code block, which should come at the end of the loop, being terminated by END_LOOP. NEXT would be the "fast" action statement causing looping without execution of the ON_REPEAT code.

```
LOOP [name]                           ! Alternates
LOOP [name] n TIMES
LOOP [name] index specification
  ...
  EXIT_LOOP [name]                    ! leaves via ON_EXIT_LOOP (if
  ...                                   any such block)
  QUIT_LOOP [name]                    ! leaves directly to next_
  ...                                   statement
  REPEAT [name]                       ! Cycles via ON_REPEAT (if
  ...                                   any such block)
  NEXT [name]                         ! Cycles via END_LOOP
  ...
ON_REPEAT [name]                      ! Optional ON_structure block
  ...
END_LOOP [name] [QUIT]                ! QUIT skips to next_statement

ON_EXIT_LOOP [name]                   ! Optional ON_structure block
  ...
DONE_EXIT_LOOP [name]                 ! Closing bracket - ending
next_statement                          "extended. loop"
```

Example of New Generalised LOOP

3.4 Yuen's example

Given the new generalised loop structure above, the example in [Yuen 84] can be expressed as:

```
LOOP N TIMES
  ...
  IF (A) THEN
      X
      EXIT_LOOP
  ENDIF
  ...
  IF (B) THEN
      Y
      EXIT_LOOP
  ENDIF
  ...
END_LOOP
ON_EXIT_LOOP
  Z
DONE_EXIT_LOOP
```

This requires neither GOTO statements, nor "invented" control variables. It seems to me that the complexity of the problem is not susceptible to further reduction.

3.5 The Subprogram/Procedure as Control Structure

I did not find (perhaps because I did not search hard enough in the literature) any reference to the subprogram/procedure body as a control structure. I believe that this is a mistake, since many of the GOTO statements used in high-quality FORTRAN code today are used for premature routine exit, via statements labelling small blocks of code that are used to set flags which will then be used by calling routines to further steer the execution sequence. This requirement should be treated in a structured fashion, and I believe that this is done in my proposal. The general form is:

```
SUBPROGRAM name [parameter list]
[declarations]
CODE                                  ! for emphasis
  ...
  EXIT_CODE                           ! returns to caller via
  ...                                   ON_EXIT_CODE
  RETURN [n]                          ! direct return to caller
  ...
END_CODE
ON_EXIT_CODE                          ! optional code block
  ...
  RETURN [n]
  ...
DONE_EXIT_CODE                        ! redundant - could be dropped
END                                   ! closes SUBPROGRAM
```

Example of New Generalised Subprogram/Procedure

Note that the QUIT_CODE statement, which would be equivalent to QUIT_LOOP, is not needed since the present RETURN [n] statement has the required meaning.

3.6 Restrictions on ON_structure Blocks

It seems perfectly legitimate to allow completely general code structures inside the ON_REPEAT, ON_EXIT_LOOP and ON_EXIT_CODE blocks. The valid range of EXIT_LOOP and QUIT_LOOP is the LOOP/END_LOOP bracket. The valid range of REPEAT is LOOP/ON_REPEAT. The valid range of NEXT is either LOOP/ON_REPEAT or LOOP/END_LOOP. The valid range of EXIT_CODE is the CODE/END_CODE bracket.

4. Final Thoughts

4.1 Impact on Compilers

There seems to be no reason why the proposed constructs would be difficult to compile, or execute inefficiently.

4.2 EXIT_LOOPs from nested LOOPS

An EXIT_LOOP [outer loop name] statement executed from inside an inner loop would presumably pass directly to the ON_EXIT_LOOP [outer loop name] structure, by-passing the ON_EXIT_LOOP [inner loop name] structure. The alternative approach appears to require too much context to be kept, and would have bad effects on efficiency.

4.3 History

After completing most of this paper I discovered the December 1974 edition of the ACM Computing Surveys, which was devoted to structured programming. In particular there is a long and interesting survey by Knuth [Knuth 74] with an interesting discussion of the history of the subject, and an extensive set of references. One reference, [Bochmann 73], does contain a short discussion of the subprogram/procedure as a control structure, which invalidates my claim in 3.5 above.

Both Bochmann and Zahn [Zahn 74] have made proposals for handling loop exits. My present proposal treats the loop exit in a rather simpler way, with one language-defined exit structure, in place of multiple user-defined structures (events for Zahn, labels for Bochmann). It also aims at a wider field, covering, in addition, the loop repeat action and the subprogram/procedure exit. However, I am more concerned that some more powerful structures should be adopted, as an aid to the humble programmer toiling to map problems on today's primitive offerings, than to claim that my ideas are the best. Let us hope that the time is ripe for some progress.

REFERENCES

[Amit 84]
A different solution for improving the readability of deeply nested IF-THEN-ELSE control structures

ACM SIGPLAN Notices 19(1), January 1984

[Bochmann 73]
Multiple exits from a loop without the GOTO

CACM 16(7), July 1973

[den Hertog et al 83]
DO-SELECT reconsidered

ACM SIGPLAN Notices 18(3), March 1983

[Dijkstra 68]
GOTO Statemetn Considered Harmful

CACM 11(3), pp 147-148, March 1968

[Hill 82]
An improvement over deeply nested IF-THEN-ELSE control structures

ACM SIGPLAN Notices 17(8), August 1982

[Knuth 74]
Structured Programming with GOTO statements

ACM Computing Surveys 6(4), December 1974

[Lakhotia 83]
An improvement over "An improvement over deeply nested IF-THEN-ELSE control structures"

ACM SIGPLAN Notices 18(5), May 1983

[Newman 83]
IF-THEN-ELSE, again

ACM SIGPLAN Notices 18(12), December 1983

[Rosenbloom 83]
Deeply nested IF-THEN-ELSE's

ACM SIGPLAN Notices 18(10), October 1983

[Yuen 83]
The programmer as navigator - a discourse on program structure

ACM SIGPLAN Notices 18(9), September 1983

[Yuen 84]
Further comments on the premature loop exit problem

ACM SIGPLAN Notices 19(1), January 1984

[Zahn 74]
A control statement for natural top-down structured programming

Symposium on Programming Languages, Paris, 1974