The Effect of Scope Delimiters on Program Comprehension

FLOYD SYKES, RAYMOND T. TILLMAN AND BEN SHNEIDERMAN
Department of Computer Science University of Maryland, College Park, MD 20740, U.S.A.

SUMMARY

Scope delimiters, such as BEGIN-END or DO-END, are used in many programming languages, but they can lengthen and clutter a program listing. This paper provides experimental evidence that ENDF or ENDWHILE statement terminators make for easier to comprehend programs than BEGIN-END pairs surrounding compound statements.

KEY WORDS Scope delimiters Programming language design Software psychology

Studies conducted at several institutions have shown that structured programming techniques improve programmer efficiency. The scope delimiting techniques used in these languages, for the most part, have not been evaluated for programmer comprehension. Some professionals have expressed the belief that the use of certain scope delimiters, such as BEGIN-END and DO-END, tends to obscure program logic:

We prefer braces because they are less obtrusive than the more common BEGIN-END or DO-END. Indentation conveys structure more clearly than large keywords.

However, there is little experimental evidence to support this conjecture.

In one experiment, different programming language structures were studied for their effects on algorithm development. Sime et al. invented three programming languages which used different scope delimiters:

(i) JUMP—using if (conditional expression) THEN GO TO (LABEL)
(ii) NEST-INE—using a not (conditional expression) and END (conditional expression)
(iii) NEST-BE—using BEGIN-END scope delimiters.

Examples of these languages are given in Figure 1

Subjects wrote recipes for cooking using these languages. Participants using NEST-BE did considerably worse than those using NEST-INE when measured by syntactic and semantic errors. Those using the JUMP language did even worse.

This pioneering experiment did not address the effect of scope delimiters on comprehension. The languages and tasks in their study were not representative of real programming languages. Other limitations were that the subjects were non-programmers and the programs were brief.

We believe that BEGIN-END and DO-END scope delimiters are 'obtrusive' and that some other form of scope delimiter would enhance comprehension.
F. SYKES, R. T. TILLMAN AND B. SHNEIDERMAN

JUMP
if hard goto L1
if juicy goto L2
chop roast stop
L2 fry stop
L1 boil stop

NEST-INE
if hard boil
not hard
if juicy fry
not juicy chop roast
and juicy
end hard

NEST-BE
if hard then
begin boil end
else
begin if juicy then
begin fry end
else
begin chop roast end
end

Figure 1. Sime's examples

HYPOTHESIS

We sought to determine the effects on program comprehension of two types of scope delimiters: BEGIN-END, as in Pascal, and ENDIF statement type, as in ADA and MAD. We expected that in comprehension tests, intermediate subjects using programs containing BEGIN-END scope delimiters would score lower than those using programs containing ENDIF (statement type) scope delimiters. We also expected that advanced programmers would score equally well on the tests regardless of the type of scope delimiters used. Finally, we believed that they would be better than intermediate programmers.

THE EXPERIMENT

Materials

To develop suitable materials to test these hypotheses, two pilot studies were conducted. The tasks required in the first pilot were too difficult. The second study, using modified materials, was successful.

A materials packet was developed which contained instructions, a consent agreement, a comprehension test, a computer program, and subjective questionnaire. All packets were identical except for the variations in the program.

The program was adapted from a textbook version of Conway's Game of Life, originally written in Pascal. This program was modified so that all FOR loops and CASE statements were rewritten as WHILE and IF-THEN-ELSE structures. Statements were also added to provide some error checking.

Three versions of the program were developed.
(a) ENDIF version used WHILE-ENDWHILE and IF-ENDIF scope delimiters. The length of the resulting program was approximately three pages.
(b) REQ-BE version used BEGIN-END scope delimiters for every simple and compound statement within a conditional structure. It was four pages long.
(c) BE version used BEGIN-END scope delimiters only for compound statements within conditional structures, following the rules of Pascal. This version was only slightly longer than the ENDIF version.

SCOPE DELIMITERS

Because layout is an important aspect of readability, it was carefully considered. The format used reflects popular, logical programming styles. The placement of scope delimiters is consistent, and allows the program versions to be as similar as possible. Subordinate structures are indented two spaces from their containing structures. BEGIN and END, which block compound statements, are aligned vertically as they are THEN and ELSE. IF and ENDIF are aligned because the ENDIF terminates the IF statement. In each case, the scope delimiter terminates a structure. The placement of the end delimiters directly under their starting delimiters reflects the syntactic structure of statements in each program version.

Examples of the formats used are given in Figure 2.

HORI = LEFT
WHILE HORI <= RT
DO
VERT = UP
WHILE VERT <= DNM
DO
BEGIN
IF OLS=[HORIZ,2]="8"
AND HORIZ < 8
OR VERT < 8
THEN
NUM = NUM + 1
END
VERT = VERT + 1
ENDWHILE
HORI = HORI + 1
ENDWHILE

ENDIF VERSION

Figure 2(a). ENDIF version

HORI = LEFT
WHILE HORI <= RT
DO
BEGIN
VERT = UP
WHILE VERT <= DNM
DO
BEGIN
IF OLS=[HORIZ,2]="8"
AND HORIZ < 8
OR VERT < 8
THEN
NUM = NUM + 1
END
VERT = VERT + 1
END
HORI = HORI + 1
END

RED-BE VERSION

Figure 2(b). REQ-BE version

HORI = LEFT
WHILE HORI <= RT
DO
BEGIN
VERT = UP
WHILE VERT <= DNM
DO
BEGIN
IF OLS=[HORIZ,2]="8"
AND HORIZ < 8
OR VERT < 8
THEN
NUM = NUM + 1
VERT = VERT + 1
END
HORI = HORI + 1
END

BE VERSION

Figure 2(c). BE version
The experimental test consisted of nine fill-in questions about the program. The test was designed so that each question built on the knowledge of the program gained from the previous question. All but one question required the participants to trace only a section of the program at a time. The final question required the subjects to trace through the complete program to determine the correct response. Most questions asked the subjects to determine values of variables after execution of program segments with initial values given in the questions.

The subjective questionnaire consisted of two parts. The first two questions concerned the subjects’ experience with the Game of Life and their understanding of the program. The remaining three questions dealt with the subjects’ on-the-job programming experience and level of training.

Subjects
Thirty-six subjects from two University of Maryland, University College upper level classes in the Organization of Programming Languages were selected to participate in the experiment. These students were familiar with the concepts of block structure and the use of scope delimiters. They had at least some experience with Pascal and PL/1.

The subjects were divided into two categories: advanced and intermediate programmers. We defined advanced programmers to have at least 2 years of on-the-job programming experience and some upper level college training in computer science. The rest of the subjects were considered to be intermediate programmers, because of the level of the course in which they were enrolled.

Administration of experiment
The experiment was administered during a scheduled class time. The subjects were given materials packets which included a version of the program. Each subject was given 2 minutes to read the instructions. They were allowed to ask questions and were asked to sign a consent agreement. The test was then administered for 25 minutes. Afterwards, the subjects answered the subjective questionnaire.

ANALYSIS
Comprehension of the programs was measured by evaluation of responses to test questions. Analysis of variance (ANOVA) was used to determine statistical significance of the scores by experience and syntax (program versions) for all groups. Mean scores and standard deviations were calculated for each group. Percentage differences of mean scores for all groups were obtained.

ANOVA revealed that the main effect of experience was significant at the $P=0.005$ level. For all groups, advanced programmers did much better than intermediate programmers (see Table 1).

The main effect of program syntax was marginally significant at the $P=0.12$ level. Group ENDIF did better than either group REQ-BE or BE.

Results were analysed by syntax group pairs using $t$-tests. In these analyses, experience yielded a significant difference at the $P=0.035$ level: advanced programmers did better in all groups. Paired $t$-tests showed that the ENDIF group did better than the REQ-BE group ($P=0.074$) and better than the BE group ($P=0.073$). The $t$-test comparing groups REQ-BE and BE showed no significance ($P=0.955$).

SCOPE DELIMITERS

<table>
<thead>
<tr>
<th>Experience</th>
<th>ENDIF</th>
<th>REQ-BE</th>
<th>BE</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced</td>
<td>11-90</td>
<td>8.86</td>
<td>8.20</td>
<td>9-29</td>
</tr>
<tr>
<td></td>
<td>(2-12)</td>
<td>(2-27)</td>
<td>(3-49)</td>
<td>(2-73)</td>
</tr>
<tr>
<td>Intermediate</td>
<td>7.83</td>
<td>5.86</td>
<td>6.33</td>
<td>6.63</td>
</tr>
<tr>
<td></td>
<td>(2-93)</td>
<td>(3-24)</td>
<td>(1-37)</td>
<td>(2-67)</td>
</tr>
<tr>
<td>Combined</td>
<td>9.27</td>
<td>7.36</td>
<td>7.18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(2-97)</td>
<td>(3-10)</td>
<td>(2-60)</td>
<td></td>
</tr>
</tbody>
</table>

Using the mean scores from Table 1, a table of percentage differences in mean values was developed (see Table II).

<table>
<thead>
<tr>
<th>Groups</th>
<th>ENDIF:REQ-BE</th>
<th>ENDIF:BE</th>
<th>REQ-BE:BE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced</td>
<td>24.2</td>
<td>34.2</td>
<td>8.1</td>
</tr>
<tr>
<td>Intermediate</td>
<td>33.6</td>
<td>23.7</td>
<td>-7.4</td>
</tr>
<tr>
<td>Combined</td>
<td>26.0</td>
<td>29.1</td>
<td>2.5</td>
</tr>
</tbody>
</table>

The table was constructed using the formula:

$$D = 100 \left( \frac{X - Y}{Y} \right)$$

$D$ is the percentage difference of the mean scores. $X$ and $Y$ are mean scores for the groups.

The differences between the means for groups ENDIF and REQ-BE and for groups ENDIF and BE indicate that participants in group ENDIF did considerably better than those in either groups REQ-BE or BE. Also, the differences between the means for groups REQ-BE and BE were comparatively small.

DISCUSSION
We designed the study around a single large program. The complexity of the tasks to be performed was limited by dividing the program into a number of logical segments. Questions were addressed to specific segments in the order in which the segments appeared in the program.

This division was desirable because it allowed us to evaluate small, closely related segments in the context of the larger program. Also, the ordered approach helped the subjects to gain an understanding of the program.

The subjects using the ENDIF version were able to answer more questions correctly than those in the other groups. Very few of the 36 subjects were able to complete the test. Of the 14 who did, 7 used ENDIF. Five subjects correctly
completed the last question which was more difficult than the other questions. Four of these used the ENDIF version of the program.

We believe that there are several reasons for these results. The reasons relate to the presence of the extra keywords BEGIN and END in program version REQ-BE and to their syntactically correct but inconsistent use in BE. The ENDIF and ENDWHILE in version ENDIF could more easily be related back to their associated starting delimiters.

The statistical evidence suggests that the use of the BEGIN and END keywords created confusion, making it more difficult for the subjects to complete the questions. These keywords do not delimit conditional statements. Rather, they block compound statements within the conditionals.

Even though there were fewer BEGIN-END blocks in program version BE, the mean scores were about the same as with the version REQ-BE. We conjecture that programmers have difficulty deciding when the BEGIN-END keywords are required. Sime et al. also believed that the rule for when 'BEGIN and END can be omitted always causes confusion'.

Figure 3 provides striking support for this. The use of BEGIN-END scope delimiters had a greater effect on program comprehension than we expected. Although advanced programmers did considerably better, the similarity between the curves for advanced and intermediate programmers suggests that both were confused by the BEGIN-END scope delimiters.

As shown by the subjective questionnaire (see Figure 4), many subjects recognized that they were confused by the BEGIN-END blocks. In fact, one subject in group BE wrote in his materials, 'I hate PASCAL!' to express the difficulties he had with the program. His score was consistent with others in his group.

When comparing the subjects' personal evaluation to their actual scores, a consistent pattern appeared. Advanced programmers most accurately assessed their understanding of the program. Intermediate programmers were not able to assess their comprehension. Some of the intermediate subjects reported that they fully understood the program. For many of these, the test scores did not support that belief.

We believe that this was because advanced programmers have greater programming skill. They are able to see the broader picture.

Figure 4 indicates the subjective estimate of comprehension by group and experience.

As this chart illustrates, the advanced programmers in the ENDIF group had more confidence in their understanding of the program than the advanced programmers in the other two groups. This is supported by their actual scores.

Group REQ-BE intermediate programmers felt just as confident in their understanding, but they did not score as well. The group BE participants were least confident of their comprehension. This is interesting, since the structure of the language used by group BE was similar to Pascal, a language familiar to all of the participants.

CONCLUSION

This study found evidence that programming languages which use the END<statement type> instead of BEGIN-END scope delimiters are easier for programmers to comprehend. The confusion introduced by the use of the extra keywords, difficulties involved in typing the BEGIN to its corresponding END delimiter and the increased program length reduces programmer efficiency.

Replication of these results with different subjects, languages and tasks would be extremely valuable. Issues such as background, style, relationship of statement delimiters to other language constructs, etc. must be considered as potential sources of bias.
If the properties of a programming language foster the writing of code which is difficult to comprehend, then the language may be more at fault than the programmer. Regardless of whether compiler writers personally prefer BEGIN-END over END statement type scope delimiters, this study indicates the need to include experimental verification of the elements of a language's design because they affect programmer performance.

REFERENCES