1978

# On the Relative Comprehensibility of Various Control Structures by Novice FORTRAN programmers

C. H. Smith

Herbert E. Dunsmore
*Purdue University*, dunsmore@cs.purdue.edu

Report Number:
79-307

# On the relative comprehensibility of various control structures by novice FORTRAN programmers,

by

H.E. Dunsmore

and

C.H. Smith

Department of Computer Science

Purdue University

West Lafayette Indiana 47907

C.S.D. TR 307

# On the relative comprehensibility of various control structures by novice FORTRAN programmers

by

C.H. Smith* and H.E. Dunsmore

Department of Computer Sciences

Purdue University

West Lafayette Indiana 47907

ABSTRACT

Two similar experiments were conducted. In the first, subjects found FORTRAN programs written with IF-THEN constructs significantly easier to comprehend than comparable programs using GOTOs. In the second experiment, programs written with GOTOs were found to be significantly easier to understand than similar programs with the GOTOs replaced by IF-THEN and WHILE-DO control structures.

Key words and phrases: structured programming, control structures, program comprehension, experimentation, human factors

CR categories: 4.22, 4.6

* Supported, in part, by NSF grant MCS 7903912.

## INTRODUCTION

The new ANS FORTRAN [Brainerd 78] (commonly referred to as FORTRAN 77) includes the IF-THEN-ELSE construction. However, structured programming advocates have been displeased by the absence of a WHILE-DO and the retention of GOTO statements [For-Word 77]. (However, the new FORTRAN DO acts a little more like a WHILE-DO and GOTOs are subject to more stringent rules than before; i.e., transfer is prohibited into either branch of an IF-THEN-ELSE [Meissner 77].) Knuth [74] has pointed out the advantage of the GOTO in implementing so-called "DO forever" loops with exits from the interior.

There are numerous existing FORTRAN implementations that include both the IF-THEN-ELSE and WHILE-DO [Moore 78; Friedman 78] and even some which have banished the GOTO. Proponents of these FORTRAN dialects are, of course, operating under the assumption that this will lead to better development and comprehension.

Little empirical evidence has been reported to support or oppose the addition of the IF-THEN-ELSE or WHILE-DO to FORTRAN. Sime, Green, and Guest [73; Green 77] found program construction simpler for naive subjects using a language in which conditional branches were IF-THEN-ELSEs rather than tests and GOTOs. However, Miller [75] using naive subjects found that the IF-THEN-ELSE was no better than tests and GOTOs for comprehensibility. Weissman's results [74], while inconclusive, suggested that pro-

grams written with structured transfer of control were easier to comprehend than unstructured programs.

We were curious as to whether the use of structured programming constructs clearly leads to more comprehensible programs. Our desire was to show empirically that such is the case. Our first experiment involved a program segment in classic FORTRAN and a modest revision that used an IF-THEN in place of a logical IF and two GOTOs. Our results suggest that the structured version is more comprehensible to our subjects.

However, our second experiment pitted a non-structured program segment against the extreme of a segment with only IF-THENS, WHILE-DOs, no GOTOs, and thus no statement numbers. Curiously, the non-structured version was more comprehensible to our subjects.

We describe the experiments, outline our results, and attempt to explain our findings in the following sections.

## The EXPERIMENTS

Two groups of novice programmers were tested at the end of a one semester programming course for their ability to comprehend programs written in FORTRAN. The paired test questions were administered as part of the student's final exam. Before presenting

the details of the experiment below we will review some characteristics of the sample populations, referred to in the sequel as Fall 78 and Spring 79.

In the Fall 78 group, 50% were freshmen, as were 42% of the Spring 79 group. 4% of the Fall 78 group had a lot of programming experience prior to the beginning of the course, 41% reported some experience, and 53% had no prior experience. The corresponding figures for the Spring 79 group were 6%, 44%, and 47% respectively. Each group was comprised primarily of science and engineering majors.

The general format of the exam given to each group was identical. The first problem consisted of several true-false type questions and the last question was a survey of prior experience, the results of which were reported above. All other questions were a mixture of program writing, comprehension, and syntax checking. The questions could be answered in any order. A time limit of two hours was imposed.

Questions 10 and 11 tested program comprehension in the Fall 78 group. In each question a program was presented and a description of its behavior was solicited. Both programs computed the Sieve of Eratosthenes [Knuth 71] (see Figure 1). The first program used three GOTOs inside a DO loop, while the second used an IF-THEN construct and a doubly-nested DO loop. The scoring, on a basis of 0 to 10, was subjective.

```
PROBLEM 10
        DIMENSION L(35)
        DATA L/35*0/
        DO 10 I=2,35
                IF (L(I).NE.0) GO TO 10
                IL=1
 1              IL=IL+I
                IF(IL.GT.35) GO TO 10
                L(IL)=I
                GO TO 1
 10     CONTINUE
        DO 20 I=1,35
                IF(L(I).EQ.0) PRINT I
 20     CONTINUE
        STOP
        END


PROBLEM 11
        INTEGER SV(35),I,J,L,MAX
        DATA SV/35*0/
        DATA MAX/35/
        J = IFIX(SQRT(FLOAT(MAX)))+1
        DO 10 I = 2,J
                IF (SV(I) .EQ. 0) THEN
                        K = I*2
                        DO 5 L = K,MAX,I
                                SV(L) = I
 5                              CONTINUE
                        ENDIF
 10             CONTINUE
        DO 20 I = 2,J
                IF (SV(I) .NE. 0) PRINT, I
20              CONTINUE
        STOP
        END
```

Figure 1
Fall 78 test programs

Questions 5 and 8 for the Spring 79 group requested the output of a given subroutine for a particular set of inputs. Hence, the scoring, on a basis of 0 to 20, was objective. Both subroutines were implementations of the SPLIT algorithm for the QUICKSORT program [Basse 78] (see Figure 2). In the second program seven GOTOs were replaced by two WHILE loops and the use of two logical variables. The two subroutines under consideration here represent a classic example of the replacement of GOTOs by logical variables [Knuth 74].

In the next section we present a statistical analysis, for both groups, of the score differences for all subjects who attempted both comprehension problems. Non-zero scores were interpreted as indicating an attempt.

## The ANALYSIS

In the Fall group 294 subjects (of the 403 who took the exam) attempted both comprehension problems, while 132 (of 306) subjects responded to both questions in the Spring 79 group. Below we present the results of the non-parametric Wilcoxon matched-pairs signed-ranks test [Siegel 56] and the parametric t-test [Roscoe 69] for both groups individually. Both tests indicate that the Fall 78 group scored significantly higher on question 11 than on question 10. Similarly, the Spring 79 scores were significantly higher on question 5 than on question 8.

```
PROBLEM 5
        SUBROUTINE STIR (X,MAX)
        INTEGER INDX, MAX, SLOT, SUB
        REAL X(MAX), SPLIT
        SLOT = 1
        SPLIT = X(SLOT)
  10    DO 20 INDX = 1,MAX
                SUB = MAX - INDX + 1
                IF (SUB .EQ. SLOT) GO TO 50
                IF(X(SUB) .GE. SPLIT) GO TO 20
                        X(SLOT) = X(SUB)
                        SLOT = SUB
                        GO TO 30
  20            CONTINUE
        GO TO 50
  30    DO 40 INDX = 1,MAX
                IF (INDX .EQ. SLOT) GO TO 50
                IF (X(INDX) .LT. SPLIT) GO TO 40
                        X(SLOT) = X(INDX)
                        SLOT = INDX
                        GO TO 10
  40            CONTINUE
  50    X(SLOT) = SPLIT
        RETURN
        END


PROBLEM 8
        SUBROUTINE MIX(A,COUNT)
        INTEGER COUNT, LCV, HOLE
        REAL TEST, A(COUNT)
        LOGICAL FLAG, DONE
        HOLE = 1
        TEST = A(HOLE)
        DONE = .FALSE.
        WHILE (.NOT. DONE) DO
                FLAG = .FALSE.
                LCV = COUNT
                WHILE (.NOT. FLAG .AND. HOLE .LT. LCV) DO
                        IF (A(LCV) .LT. TEST) THEN
                                A(HOLE) = A (LCV)
                                HOLE = LCV
                                FLAG = .TRUE.
                                ENDIF
                        LCV = LCV - 1
                        ENDWHILE
                DONE = .NOT. FLAG
                FLAG = .FALSE.
                LCV = 1
                WHILE (.NOT. FLAG .AND. HOLE .GT. LCV) DO
                        IF (A(LCV) .GT. TEST) THEN
                                A(HOLE) = A(LCV)
                                HOLE = LCV
                                FLAG = .TRUE.
                                ENDIF
                        LCV = LCV + 1
                        ENDWHILE
                DONE = .NOT. FLAG
                ENDWHILE
        A(HOLE) = TEST
        RETURN
        END
```

The Wilcoxon test is applicable when both the sign and the magnitude of the differences are meaningful. For each subject the difference was calculated as the score on question 10 minus the score on question 11 (Fall 78) or the score on question 5 minus the score on question 8 (Spring 79). Typically, zero differences are omitted and the sample size is reduced accordingly. Our data showed 45% zero differences for the Fall 78 group and 23% zero differences for the Spring 79 group. Hence, the more conservative approach of considering the zero differences to be half negative and half positive was taken. The results, summarized in Figure 3, indicate that the Fall 78 group scored higher on question 11 than on qustion 10 and the Spring 79 group scored higher on question 5 than on question 8. Both differences were significant at the .01 level.

The t-test for the differences in related measures is applicable when the differences are normally distributed or when the sample size is larger than 30. Both samples are much larger than 30 and as evidenced by Figure 4, both distributions seem approximately normal. Figure 5 indicates that the observed differences were significant at the .01 level.

## CONCLUSIONS

The programs in questions 10 and 11 are functionally equivalent. Furthermore, the two programs have many other syn-

FALL 78

| | |
|---|---|
| 132 | zero differences |
| 98 | negative differences |
| 64 | positive differences |
| 294 | sample size |

$T_+ = 12106$

$T_0 = 8778$

$T = T_+ + 1/2\ T_0$

$z = -3.55$ alpha < .01

SPRING 79

| | |
|---|---|
| 31 | zero differences |
| 34 | negative differneces |
| 67 | positive differences |
| 132 | sample size |

$T_- = 2795$

$T_0 = 496$

$T = T_- + 1/2\ T_0$

$z = -3.06$ alpha < .01

---

Legend:

$T_0$ sum of the zero ranks

$T_-$ sum of the negative ranks

$T_+$ sum of the positive ranks

Figure 3
Results of the Wilcoxon matched-pair signed-rank test

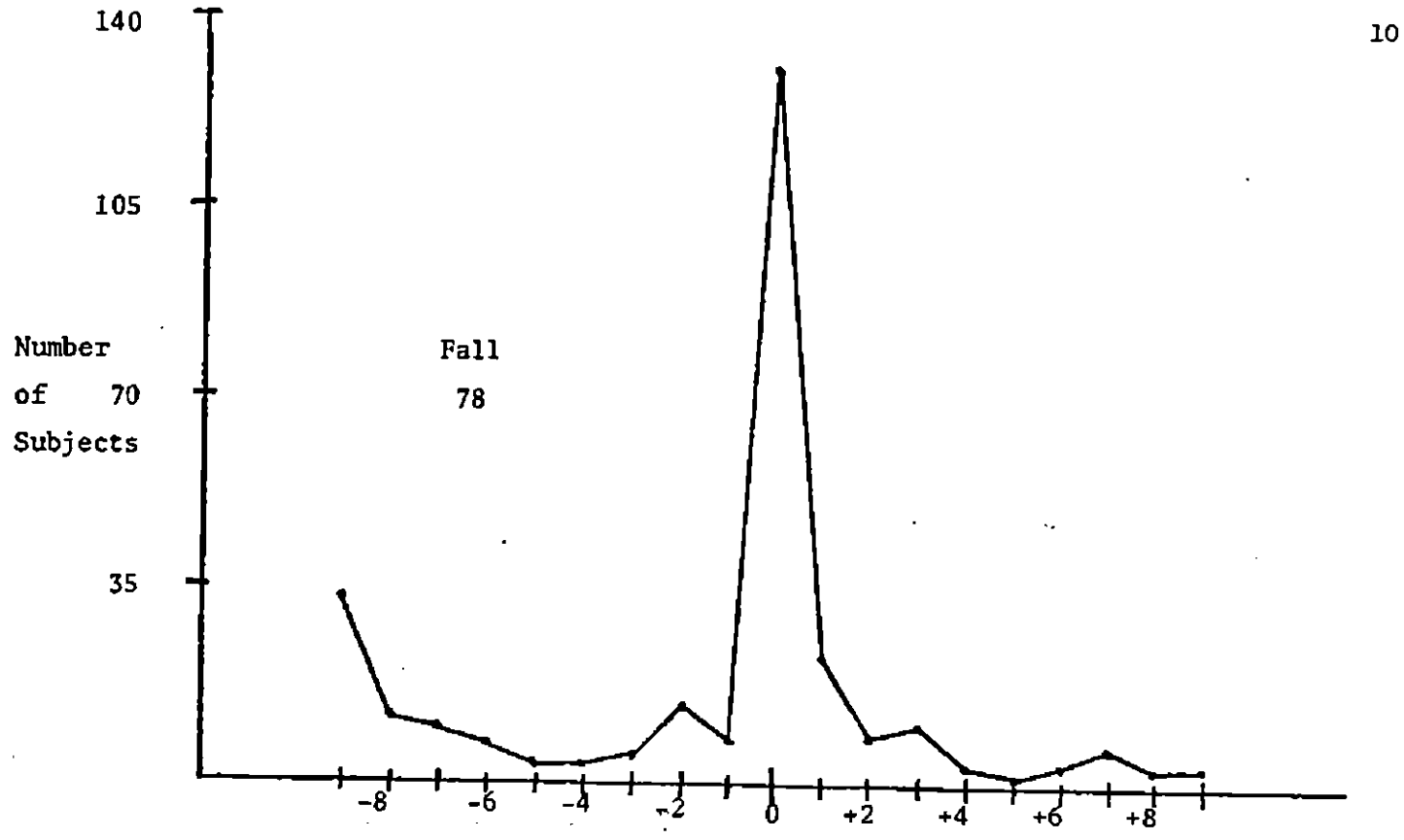


Figure 4
Distribution of the differences

FALL 78

| | |
|---|---|
| sample size | 294 |
| mean | -1.2925 |
| standard deviation | 4.22 |
| t | -5.25 |
| alpha | <.01 |

SPRING 79

| | |
|---|---|
| sample size | 132 |
| mean | 0.9679 |
| standard deviation | 4.27 |
| t | 2.61 |
| alpha | <.01 |

Figure 5

Results of the t-test for difference scores

tactic features in common (see Figure 6). The significantly higher scores on question 11 can be explained either by the increased comprehensibility offered by the IF-THEN statements over GOTOs, or by a learning effect of subjects answering question 10 first. No restriction was placed on which order the questions were to be answered. On examinations students are known to answer questions out of order. We have no reason to suspect that our subjects were atypical. In light of the degree of significance between the scores on questions 10 and 11, it seems that for this small program segment, the IF-THEN control structure is preferable to the GOTO for comprehension by novice programmers.

The interpretation of the results of the Spring 79 experiment is more controversial. This group scored higher on the first question, so apparently learning effects are irrelevant. More importantly, the two test subroutines varied greatly in length and other syntactic measures (see Figure 6). We therefore conclude that either GOTOs are preferable from the standpoint of novice programmers, to a combination of WHILE and IF-THEN constructs or that comprehension by novice FORTRAN programmers is more influenced by program length than by control structure characteristics.

Our results seem to support the FORTRAN 77 decision to include the IF-THEN-ELSE construction in the standard while omitting the WHILE-DO. However, we realize that preliminary evidence of this sort is insufficient as a confirmation of such decisions.

| Syntactic features | | | | |
|---|---|---|---|---|
| number of | Q 11 | Q 10 | Q 5 | Q 8 |
| statements | 17 | 15 | 24 | 34 |
| statement labels | 3 | 3 | 5 | 0 |
| structured GOTOs | 0 | 2 | 3 | 0 |
| non structured GOTOs | 0 | 1 | 4 | 0 |
| FORTRAN DO-loops | 3 | 2 | 2 | 0 |
| WHILE-DOs | 0 | 0 | 0 | 3 |
| FORTRAN logical IFs | 1 | 3 | 4 | 0 |
| IF-THENs | 1 | 0 | 0 | 2 |
| variables and constants | 9 | 7 | 7 | 10 |
| references to variables and constants | 26 | 23 | 40 | 57 |

Figure 6
Syntactic features of test programs

We encourage other researchers to conduct and to report the results of similar experiments.

ACKNOWLEDGEMENTS

We wish to thank G. Campbell for his advice on the statistical analyses. D. Sorge and his staff were instrumental in helping us collect the data. Course instructors D. Boss, R. Brown, N. DiMasi, J. Moe, D. Reed, E.G. Sewell, and C. Smith were most cooperative in allowing us to insert suitable comprehension questions into the final exam given to their students. D. Edwards and D. Dietrich assisted us with the data analysis. Finally, we wish to thank all the students who participated in this study. Computer time was provided by the Department of Computer Science, Purdue University, and by the Purdue University Computing Center.

## REFERENCES


Basse, S., Computer Algorithms: Introduction to Design and Analysis , Addison Wesley, 1978.

Brainerd, W., "FORTRAN 77," Communications of the ACM , Vol 21, October 1978, pp 806-820.

For-Word, Fortran Development Newsletter, ACM SIGPLAN NOTICES , Vol 12, April 1977, pp 21-30.

Friedman, F. and Koffman, E., "Teaching problem solving and structured programming in FORTRAN," Computers and Education , Vol 2, 1978, pp 235-245.

Green, T., "Conditional program statements and their comprehensibility to professional programmers," Journal of Occupational Psychology , Vol 50, 1977, pp 93-109.

Knuth, D., The Art of Computer Programming , Vol 2, Addison Wesley, 1971.

Knuth, D., "Structured programming with go to statements," ACM Computing Surveys , Vol 6, December 1974, pp 261-301.

Meissner, L., "Fortran 77," ACM SIGPLAN NOTICES , Vol 12, January 1977, pp 93-94.

Miller, L., "Naive programmer problems with specification of transfer-of-control," Proceedings of the National Computer Conference , 1975, pp 675-663.

Moore, J. and Makela L., Structured Fortran with Watfiv , Prentice-Hall, 1978.

Roscoe, J., Fundamental Research Statistics for the Behavioral Sciences , Holt, Rinehart, and Winston, 1969.

Siegel, S., Nonparametric Statistics for the Behavioral Sciences , McGraw-Hill, 1956.

Sime, M., Green, T., and Guest, D., "Psychological evaluation of two conditional constructions used in computer languages," International Journal of Man-Machine Studies , Vol 5, 1973, pp 105-113.

Weissman, L. "Psychological complexity of computer programs: an experimental methodology," ACM SIGPLAN NOTICES , Vol 9, June 1974, pp 25-36.