

6-1-2021

The Cloze Procedure as a Measure of Program Understanding.

Ibrahim Eissa

Institute of Statistical Studies and Research, Cairo University, Egypt.

Said Selim

Institute of Statistical Studies and Research, Cairo University, Egypt.

Follow this and additional works at: <https://mej.researchcommons.org/home>

Recommended Citation

Eissa, Ibrahim and Selim, Said (2021) "The Cloze Procedure as a Measure of Program Understanding.," *Mansoura Engineering Journal*: Vol. 10 : Iss. 1 , Article 3.

Available at: <https://doi.org/10.21608/bfemu.2021.176856>

This Original Study is brought to you for free and open access by Mansoura Engineering Journal. It has been accepted for inclusion in Mansoura Engineering Journal by an authorized editor of Mansoura Engineering Journal. For more information, please contact mej@mans.edu.eg.

THE CLOZE PROCEDURE AS A MEASURE
OF PROGRAM UNDERSTANDING

BY

Ibrahim F. Eissa* AND Said. M. Selim**

ABSTRACT

The process of program understanding plays an important role in testing, and maintenance phases of software life cycle. There have been numerous investigations of the influence of various aspects of a program on program understanding, using many different measures of understanding. Some of the different measures include time to find a bug, a comprehension quiz, ability to reproduce a functionally equivalent program without notes, time to perform modification. All of these measures have some limitations.

A new measure called CLOZE procedure, which has less limitations as well as other advantages such as ease of administration and grading, is investigated in this paper.

The paper reports on a controlled experiment that compared CLOZE procedure and the time to find a bug as measuring program understanding.

The experiment results indicated that the CLOZE procedure measure is a fair measure for program understanding. It may best be used in educational environments where computerizing test construction, as well as grading, is highly appreciated.

1. Introduction

The process of program understanding plays an important role in testing and maintenance phases of the software life cycle. If one cannot understand a program, how can he expect to test and maintain it? Software testing and maintenance phases constitute 45% of the total cost of the software development process [7].

* & ** Institute of Statistical Studies and Research,
Cairo University, Egypt.

Therefore, identifying any factors that contribute to decreasing this cost is economically sound.

There have been numerous investigations of the influence of various aspects of a program on program understanding. A partial list of these includes: modularization [14], comments [14,11], indenting [2], structured coding [8], mnemonic variable names [3], program length [3], flowcharts [13], documentation [10], and control flow [12]. Almost all of these investigations involved controlled experimentations.

During experimentation, some sort of a measure is required to evaluate the degree of program understanding. A wide variety of program understanding measures were used in such investigations: comprehension quiz [11, 14], time to perform a modification [2], time to locate a bug [2], Halstead's E (programming effort) [6], and reproduction of functionally equivalent program without notes [3].

All of these measures have limitations such as inability to measure both low-and high-level understanding, difficulty of administering and objectively grading [1]. Due to these limitations, Cook [1] proposed a new measure; the CLOZE procedure measure. The word "CLOZE" refers to the human tendency to complete a familiar but not quite finished pattern. Cook investigated the use of the CLOZE procedure as a measure of program understanding and compared it to the comprehensive quiz [1]. His results showed that the scores of both measures were identical.

It is well-known that test construction is an onerous task for instructors. Cook [1] indicated that the CLOZE procedure alleviates not only test construction but also the grading burden; both can be computerized. Due to these reasons, the authors were motivated to carry a further investigation for the CLOZE procedure measure.

This paper reports on a controlled experiment that compared the CLOZE procedure and debugging task (time to locate bug) as measuring program understanding. In a CLOZE procedure, the subjects are presented a program listing with some of the program tokens (operands, operators, reserved words, single parenthesis or brackets, etc.) replaced with blanks and are required to fill in the blanks. Our choice for the debugging tasks is probably the most realistic choice. This is due to the fact that debugging is a natural activity in software life cycle.

In the remaining sections, we describe the experiment we conducted and analyze its results.

2. The Experiment: CLOZE Procedure vs, Debugging Task

2.1 Subjects and Experiment Objectives

The subjects were students who had two semester courses in PASCAL and one semester course in FORTRAN. There were 96 students tested, voluntarily, to measure their ability to comprehend programs written in FORTRAN. All the students were at the same academic level; junior computer science.

The experimental objective was to compare the CLOZE measure vs. the debug task; also, to study the effect of bug type on the time to locate the bug.

2.2 Material

Two FORTRAN Programs with the same complexity, according to McCabe complexity measure [9], were used. The first program is a crosstabulation program (TABL). The other program (STAT) is to calculate the totals, averages, minima, and maxima values of a set of observations. The correct programs listing are shown in Appendix A, Pages A1 and A2.

Two forms for each program were used in experimentation: a CLOZE form and a debug form. The CLOZE form of each program had 6 missing tokens, shown on pages A3 and A4. The debug forms were exactly the correct program text, with only a single bug in it.

To measure the effect of bug type on the time to locate it, three types of bugs were chosen: Assignment, Iteration, and Control. This choice was influenced by the basic components of structured control structure [4]. It should be noted that the missing tokens in the CLOZE forms were selected on the same basis. Indeed, the tokens have been selected to reflect and include the chosen bugs.

Having chosen the bugs this way, we were faced by, the problem of whether to include more than one bug in the debug form or not. The inclusion of more than one bug has the serious drawback of possibly confusing the students when deciding which bug caused the error in the program. Therefore, the decision was made to include only a single bug in each debug form, ending with 3 variations of that form for each program. This is to

keep the reflection of bug type on the debug time. The selected bugs are shown with the program listings, pages A1 and A2. During experimentation, the bugs replaced their correct version.

2.3 Procedure

The experiment was conducted for all subjects at the same time. Each subject was asked to answer a package of two parts: a CLOZE part and a debug part.

The CLOZE part consisted of a CLOZE form of either program preceded by a description of the CLOZE procedure and illustrative example. The debug part consisted of one of the three variations of the other program's debug forms. Both forms in each part were preceded by a complete documentation and a sample input, together with its correct output. The aim was that the subject would use this information to locate the bug or to fill in the missing tokens.

The ordering of forms within the distributed packages was arranged so that:

- a. Half the students answered the CLOZE part first followed by the debug part. The other half answered in the reverse order.
- b. For each form (or variation of a form), half of its distributed numbers was answered first; the other was answered next.

Due to this ordering, we ended up with 12 different (order-wise) packages as Table A illustrates. Appendix B includes one of those packages. During experimentation, the test materials were distributed randomly. The aim of this ordering process was to eliminate the mental exhaustion phenomenon from being an effective factor in the experiment.

The time allocated for each part was 20 minutes. Students were told to raise their hands when thinking that they found the bug and they would be notified whether or not they found the right bug. If not, and there was more left, they resumed again. There was no mention about either the type or the place of the bug.

TABLE A

Packages	TABL Forms			STAT Forms				
	CLOZE	1	2	3	CLOZE	1	2	3
1	a					b		
2	a						b	
3	a							b
4	b					a		
5	b						a	
6	b							a
7		b			a			
8			b		a			
9				b	a			
10		a			b			
11			a		b			
12				a	b			

- a means answered first, followed by b (on the same line).
- Each non-empty cell of the table represents 8 students.

3. Results and Its Analysis

The results of the experiment are summarized in Table 1-7. Tables 1-5 are the raw data; Tables 6 and 7 are deduced from Tables 1-5. In the tables, I means iteration, C means Conditiona, and A means Assignment.

3.1 Comparing the Two Measures

Table 2 of the TABL program shows that the number of students who scored in the CLOZE form was 49; among those only 13 found the bug in the debug form. The corresponding numbers for the STAT program were 47 and 14, respectively (Table 4). This contrast in numbers indicates a difference between the two measures. This raises the following questions: Why this difference? Is it total or partial difference?

Of course, the real and concrete answer to these questions is beyond the limitations of a single experiment; too much experimentation is required to have a solid answer. However, within the limited results of our experiment, we related this difference to the fact that in the debug measure, the students were positioned in a black or white situation. Their answers were either zero or full mark; and there was no credit for any partial effort.

TABL Program Results

Bug Type	No. of Correct Answers	Time	Avr. Time
I	6	14,6,12,15,14,16	12.83
C	4	13,21,15,11	15.00
A	3	19,17,22	19.33

Table-1. Debug Scores.

Subject Classification	No. of Subjects	Scores (out of 6)	Avr. Score
Found the Bug	13	6,4,4,3,4,6,4,5,6,3,3,5,6	4.54
Did Not Find the Bug	36	3,0,5,3,4,5,4,5,1,6,3,3,5,6,4,5, 2,5,2,4,6,3,3,5,4,5,4,4,3,6,6,3, 6,2,4	3.97

Table-2. Cloze Scores.

STAT-Program Results

Bug Type	No. of Correct Answers	Time	Avr. Time
I	7	9,7,13,10,12,14,8	10.43
C	3	14,17,15	15.33
A	4	18,20,16,18	18.00

Table-3. Debug Scores.

Subject Classification	No. of Subjects	Scores (out of 6)	Avr. Score
Found the Bug	14	5,4,3,6,2,3,5,6,4,6,2,5,2,4	4.07
Did Not Find the Bug	33	2,3,2,4,3,6,2,3,5,0,4,3,1,4,5,0,5, 1,4,5,5,5,4,3,6,2,3,4,5,4,2,6,4	3.48

Table-4. Cloze Scores.

TABL		STAT	
Token Type	How Many Times Missed	Token Type	How Many Times Missed
I	10	A	17
A	8	I	6
C	11	C	21
A	24	I	18
C	24	C	4
A	27	C	19

Table-5.

- Missed tokens are those which have not been filled or wrong.

Token Type	Token Count	How Many Times Missed	Average
I	3	34	11.3
C	5	79	15.8
A	4	76	19.0

Table-6. (is an aggregation of Table-5.)

The idea behind this table is that the larger the number of a missing token the longer time it would take if it were filled correctly. Therefore the average column represents time.

Bug Type	Average		Total Average
	TABL	STAT	
I	12.83	10.43	11.63
C	15.00	15.33	15.17
A	19.33	18.00	18.67

Table-7. (constructed from Tables 1 and 2.)

A second reason is that the students who found the bug may be better than those who did not. Indeed, this turned out to be true from the following:

- a. From Tables 2 and 4, we can see that the average CLOZE score for the students who found the bug is higher than for those who did not.
- b. Nevertheless, this average is still higher if we compare the students who found the bug vs. the total number of students (those who found the bug and those who did not). This is shown in the following table:

TABLE B

Program	Average CLOZE Score	
	LABL	STAT
Student found bug	5.54	4.07
Total number of students	4.12	3.66

Having justified the difference, a similarity between the two measures can be seen by comparing the last columns in Tables 6 and 7. Here the two measures agree that the Iteration bugs are the easiest to locate and the Assignment bugs are the most difficult. In between lies the Conditional bugs. However, this similarity by no means implies total similarity.

3.2 The CLOZE Procedure as Stand Alone Measure

The analysis of the results indicated a fair degree of subjectivity in using the CLOZE procedure as a measure of program understanding for the following reasons:

a. Distribution of Students:

Figure 1 shows the distribution of the 96 students against the CLOZE scores. Using the method of normal approximation [5], the percentage of students whose scores are within one standard deviation (SD) is 65.62%, and within two SD is 96.87%. The corresponding figures for the proper normal curve are 68% and 95%. Therefore, it is clear that the distribution of the students is nearly normal.

b. The CLOZE measure over different programs:

The fairness of the CLOZE measure over different programs is also justifiable. This is clear by working out the relative difference in CLOZE score averages for our two programs. From Table B these are:

$$\text{For TABL: } 100 (4.54 - 4.12)/4.54 = 9.25\%$$

$$\text{For STAT: } 100 (4.07 - 3.66)/4.12 = 10.07\%$$

which are very close figures.

4. Comments and Conclusion

It has been noticed during experimentation that the allowable time (20 minutes) was enough for the CLOZE part. However, there was frustration for more time in the debug part. Had this time extension been allowed would the results have been different? Of course, the answer needs further investigation.

The analysis of the results showed some differences as well as similarities between the two measures. Also, it showed the fairness of the CLOZE procedure measure as a stand alone measure. Consequently, the authors feel that the CLOZE measure may be best used in educational environments. This serves twofold: first, crediting any partial effort; secondly, by computerizing the process of test construction and grading, the burden on instructors will be decreased.

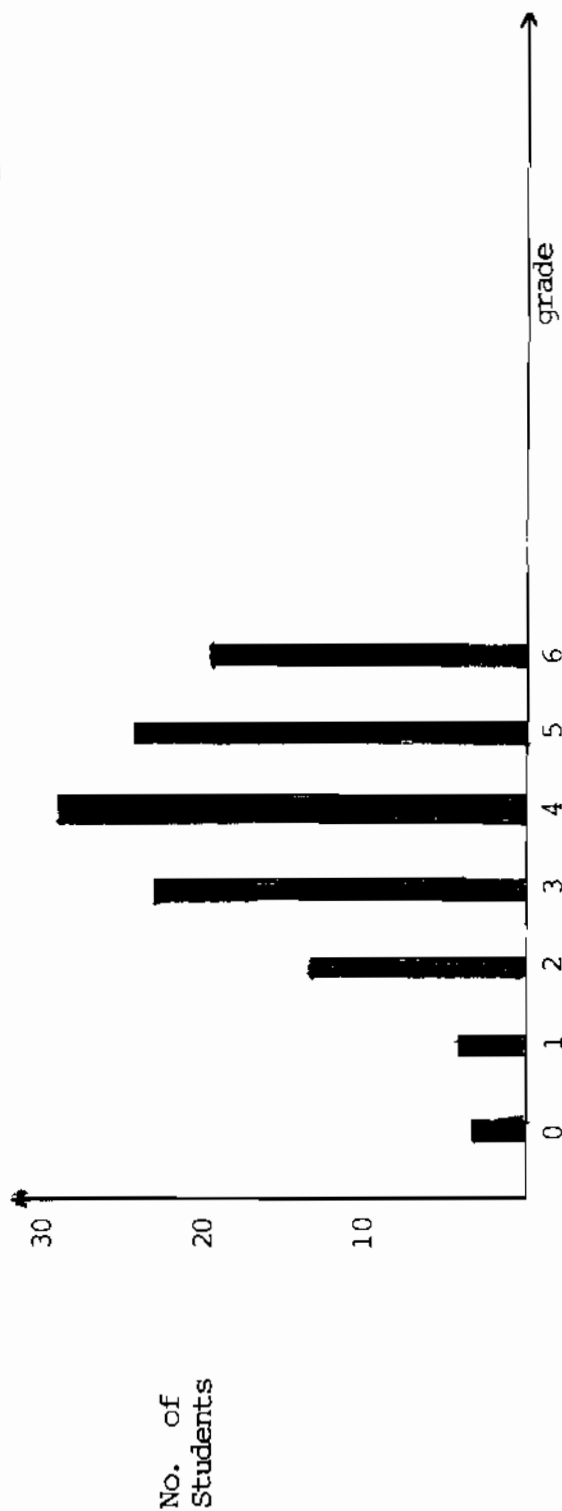
Both measures indicated that the Assignment bugs are the most difficult to locate while the iteration is the easiest; in between lies the Conditional. Whether this is true or not again needs further experimentation.

Almost all experimental measures proposed in the literature provide a measure for program understanding to some extent. But, the answers to the following questions are not clear:

- . What are the criteria for selecting an experimental measure for program understanding?
- . What are the types of questions to be included in the testing method?

These questions need further investigation; they are open research problems in literature.

grade	No. of Students
0	3
1	3
2	11
3	14
4	24
5	20
6	16



REFERENCES

- [1] Cook, C.W. Breger, and D. Foote. A preliminary investigation of the use of the CLOZE procedure as a measure of program understanding. Information Processing and Management (1984), Vol. 20, 199-208.
- [2] Curtis, B., S. B. Sheppard, and P. Milliman. Third Time charm: Stronger replication of the ability of software complexity metrics to predict programmer performance. Proceedings of Fourth International Conference on Software Engineering, Munich, Germany (September, 1979), 356-360.
- [3] Curtis, B., S. B. Sheppard, P. Milliman, M.A. Borst, and T. Love. Measuring the psychological complexity of software maintenance tasks with Halsted and McCabe metrics. IEEE Trans, Software Eng., Vol. SE-5 (1979), 96-104.
- [4] Dijkstra, E. W. Notes on structured programming, in Structured Programming by Dahl, Dijkstra, and Hoane. Academic Press, N.Y., 1072.
- [5] Freedman, D., R. Pisani, and R. Purves. Statistics, Chapter 5. Norton, New York, 1978.
- [6] Gordon, R. Measuring improvements in program clarity. IEEE Trans. Software Eng., Vol. SE05 (1979), 79-90.
- [7] Lewis. T. Software engineering analysis and verification. Reston Publishing Company, Inc., 1982.
- [8] Love, T. An experimental investigation of the effects of program structure on program understanding. ACM SIGPLAN Notices, 10 (March, 1977, 105-113.
- [9] McCabe, T. J. A complexity measure. IEEE Trans. Software Eng., Vol. SE-2 (1976), 308-320.
- [10] Sheppard, S. B., E. Kruesi, and B. Curtis. The effects of symbology and Spatial arrangement on the comprehension of software specifications. Proceedings of Fifth International Conference on Software Engineering, San Diego, California (March, 1981), 207-214.
- [11] Sheppard, S. B., M. A. Borst, and B. Curtis, Predicting programmer ability to understand and modify software. Proceedings of Symposium on Human Factors and Computer Science. Washington, D.C. (June, 1978), 115-135.

- [12] Shneiderman, B. Control flow and data structures documentation: Two experiments. *Comm. ACM*, 25 (1982), 56-63.
- [13] Shneiderman, B., R. Mayer, D. McKay, and P. Heller. Experimental investigation of the utility of detailed flowcharts in programming. *Comm, ACM*, 20 (1977), 373-381.
- [14] Woodfield, S. N., and H. E. Dunsmore. The effect of modularization and comments on program comprehension. *Proceedings of Fifth International Conference on Software Engineering, San Diego, California* (1981), 215-223.


Appendix-A

```

PROGRAM TABL(INPUT,OUTPUT,TAPE5=OUTPUT)
INTEGER AGE,W,A
DIMENSION A(5,6),W(30),AGE(30)
READ(*,100) (AGE(I),I=1,30)
READ(*,100) (W(I),I=1,30)
100 FORMAT(30I2)
WRITE(5,101) (AGE(I),I=1,30)
WRITE(5,101) (W(I),I=1,30)
101 FORMAT(30I3)
IA=4
IW=5
N=IA+1
M=IW+1
DO 1 I=1,N
DO 1 J=1,M
1 A(I,J)=0
DO 10 K=1,30
IF(AGE(K) .LE. 0)AGE(K)=1
IF(AGE(K) .GT. 15)AGE(K)=16
I=((AGE(K)-1)/5)+1
IF(W(K) .LT. 20)W(K)=20
IF(W(K) .GT. 60)W(K)=60
J=(W(K)/10)-1
A(I,J)=A(I,J)+1
A(N,J)=A(N,J)+1
A(I,M)=A(I,M)+1
10 A(N,M)=A(N,M)+1
DO 15 I=1,N
15 WRITE(5,101) (A(I,J),J=1,M)
STOP
END

```

BUGS



```

DO 10 K=1,15
IF(AGE(K) .GE.15)AGE(K)=16
I=((AGE(K)+1)/5)+1

```

```

-1 0 1 3 4 3 5 2 6 7 7 8 8 9 9 10 10 14 11 12 15 13 15 16 20
16 19 17 20 18
29 1 30 40 41 50 52 60 2 3 31 32 41 43 55 54 61 9 35 37 44 56 62 7 38
49 57 59 60 45
2 1 2 2 1 8
2 2 2 2 1 9
1 2 1 1 1 6
1 1 2 2 1 7
6 6 7 7 4 30

```

TABL Correct Listing and Bugs

E. 42 I. F. EISSA & S. V. SELIM

```

PROGRAM STAT(INPUT,OUTPUT,TAPE1=OUTPUT)
INTEGER S,SCNT
DIMENSION A(15),S(5),TOTAL(3),AVER(3),VMIN(3),VMAX(3)
READ(*,100) (A(I),I=1,15)
READ(*,101) (S(I),I=1,5)
100 FORMAT(15F3.1)
101 FORMAT(5I2)
WRITE(1,101) (S(I),I=1,5)
WRITE(1,102) (A(I),I=1,15)
102 FORMAT(15F4.1)
NV=3
NO=5
DO 1 K=1,NV
TOTAL(K)=0.0
AVER(K)=0.0
VMIN(K)=1.0E10
1 VMAX(K)=- 1.0E10
SCNT=0
DO 7 J=1,NO
IJ=J-NO
IF(S(J) .EQ. 0) GO TO 7
SCNT=SCNT+1
DO 6 I=1,NV
IJ=IJ+NO
TOTAL(I)=TOTAL(I)+A(IJ)
IF(A(IJ) .LT. VMIN(I))VMIN(I)=A(IJ)
IF(A(IJ) .GT. VMAX(I))VMAX(I)=A(IJ)
6 CONTINUE
7 CONTINUE
IF(SCNT .EQ. 0) GO TO 13
DO 10 I=1,NV
10 AVER(I)=TOTAL(I)/SCNT
15 DO 20 I=1,3
20 WRITE(1,103) (TOTAL(I),AVER(I),VMIN(I),VMAX(I))
103 FORMAT(4F6.1)
STOP
END

```

BUGS



```

VMIN(K)=-11.0E10
VMAX(K)=1.0E10

```

```
DO 7 J=1,NV
```

```
IF(S(J) .EQ. 0) GO TP 5
```

```

1 2 0 4 5
4.0 6.0 7.5 4.5 6.5 8.0 5.0 7.0 8.5 5.5 7.5 9.5 6.0 8.0 9.5
21.0 5.3 4.0 6.5
27.0 6.8 5.0 8.5
34.5 8.6 7.5 9.5

```

STAT Correct Listing and Bugs

```

PROGRAM TABL(INPUT,OUTPUT,TAPES=OUTPUT)
INTEGER AGE,W,A
DIMENSION A(5,6),W(30),AGE(30)
READ(*,100) (AGE(I),I=1,30)
READ(*,100) (W(I),I=1,30)
100 FORMAT(30I2)
WRITE(5,101) (AGE(I),I=1,30)
WRITE(5,101) (W(I),I=1,30)
101 FORMAT(30I3)
IA=4
IW=5
N=IA+1
M=IW+1
DO 1 I=1,N
DO 1 J=I,M
1 A(I,J)=0
DO 10 K=1,
IF(AGE(K) .LE. 0) AGE(K)=
IF(AGE(K) ----- 15) AGE(K)=16
I=((AGE(K) ----- 1)/5)+1
IF(W(K) .LT. 20) W(K)=20
IF(W(K) ----- 60) W(K)=60
J=(W(K)/10) ----- 1
A(I,J)=A(I,J)+1
A(N,J)=A(N,J)+1
A(I,M)=A(I,M)+1
10 A(N,M)=A(N,M)+1
DO 15 I=1,N
15 WRITE(5,101) (A(I,J),J=1,M)
STOP
END

```

TABL CLOZE form

E. 44 I. F. EISSA & S. M. SELIM

```

PROGRAM STAT(INPUT,OUTPUT,TABLE1=OUTPUT)
INTEGER S,SCNT
DIMENSION A(15),S(5),TOTAL(3),AVER(3),VMIN(3),VMAX(3)
READ(*,100) (A(I),I=1,15)
READ(*,101) (S(I),I=1,5)
100 FORMAT(15F3.1)
101 FORMAT(5I2)
WRITE(1,101) (S(I),I=1,5)
WRITE(1,102) (A(I),I=1,15)
102 FORMAT(15F4.1)
NV=3
NO=5
DO 1 K=1,NV
TOTAL(K)=0.0
AVER(K)=0.0
VMIN(K)=1.0E10
1 VMAX(K)=-----
SCNT=0
DO 7 J=1,-----
IJ=J-NO
IF(S(J) .EQ. 0) GO TO -----
SCNT=SCNT+1
DO 6 I=-----
IJ=IJ+NO
TOTAL(I)=TOTAL(I)+A(IJ)
IF(A(IJ) ----- VMIN(I))VMIN(I)=A(IJ)
IF(A(IJ) .GT. VMAX(I))VMAX(I)=A(IJ)
6 CONTINUE
7 CONTINUE
IF(SCNT ----- 0) GO TO 15
DO 10 I=1,NV
10 AVER(I)=TOTAL(I)/SCNT
15 DO 20 I=1,3
30 WRITE(1,103) TOTAL(I),AVER(I),VMIN(I),VMAX(I)
103 FORMAT(4F6.1)
STOP
END

```

STAT CLOZE form

Appendix B
READ THIS ONLY

1. DO NOT EXAMINE THE OTHER PAGES OF THIS HANDOUT UNTIL INSTRUCTED TO DO SO.
2. THIS IS AN EXPERIMENT ONLY. A SERIOUS EFFORT IS EXPECTED. YOUR PERFORMANCE WILL NOT AFFECT YOUR GRADE WHATSOEVER.
3. YOUR PARTICIPATION IN THIS EXPERIMENT IS VOLUNTARY. YOU ARE FREE TO WITHDRAW AT ANY TIME.

For this experiment there is a practice problem followed by the actual problem. The practice problem is to acquaint you with the testing procedure being used in this experiment. After completing the practice problem do the actual problem.

PRACTICE PROBLEM

A token is a simple variable identifier, an array identifier, a procedure or function name, a subscript, a constant or an operator (logical or arithmetic). Certain of the tokens in the following pascal program have been deleted and replaced with an underline. Your task is to fill in the deleted tokens.

```

PROGRAM GCD (INPUT,OUTPUT, TAPE1 = OUTPUT)
  INTEGER R,M,N
  READ (*,100) M, N
100  FORMAT (2I3)
  50  R = M - (M/ -----) * N
      M = N
      ----- = R
      IF(R----- 0) GO TO -----
      WRITE (1,101) M
101  FORMAT (' GREATEST COMMON DIVISOR IS',I3)
      STOP
      END

```

So you can compare your answer, the program with the correct answers appears on the next page.

E. 46 I. F. EISSA & S. M. SELIM

```
PROGRAM GCD (INPUT, OUTPUT, TAPE1 = OUTPUT)
  INTEGER R,M,N
  READ(*,1000) M, N
100  FORMAT (2I3)
  50  R = M - (M / N) * N
      M = N
      N = R
      IF (R .GT. 0) GO TO 50
      WRITE (1,101) M
101  FORMAT (' GREATEST COMMON DIVISOR IS', I3)
      STOP
      END
```

On the following page is the actual problem. You are to fill in the missing tokens the best you can. You will have a maximum of 20 minutes to complete this part of the experiment.

NOTE: The program is preceded by its description.

DOCUMENTATION

Purpose of Program:

To corss tabulate the age against the weight.
That is the number of subjects in each age and weight interval.

Data:

AGE(30) is input vector containing ages

W(30) is input vector containing weights

A(5,6) is output matrix contains the results of cross tabulations

The four age intervals are:

age ≤ 5 , $5 < \text{age} \leq 10$, $10 < \text{age} \leq 15$, age > 15

The five weight intervals are:

Weight < 30 , $30 \leq \text{weight} < 40$, $40 \leq \text{weight} < 50$,
 $50 \leq \text{weight} < 60$, weight ≥ 60 .

INPUT DATA

-100010304030502060707080809091010141112151351620816191720
2901304015052600203313241435554610935374456620738454957960

CORRECT OUTPUT

WEIGHT \ AGE	<30	30<=W 40	40<=W<50	50<=W<60	W>=60	TOTAL
A <=5	2	1	2	2	1	8
5<A<=10	2	2	2	2	1	9
10<A<=15	1	2	1	1	1	8
A > 15	1	1	2	2	1	7
TOTAL	6	6	7	7	4	30

E. 48 I. F. EISSA & S. M. SELIM

```

PROGRAM TABLE (INPUT, OUTPUT, TAPE5=OUTPUT)
INTEGER AGE,W,A
DIMENSION A(5,6),W(30),AGE(30)
READ(*,100) (AGE(I), I=1,30)
READ(*,100) (W(I), I=1,30)
100 FORMAT(30I2)
WRITE(5,101) (AGE(I), I=1,30)
WRITE(5,101) (W(I), I=1,30)
101 FORMAT(30I3)
IA=4
XW=5
N=IA+1
M=W+1
DO 1 I=1,N
DO 1 J=1,M
1 A(I,J)=0
DO 10 K=1,
IF (AGE(K) .LE. 0) AGE(K)=
IF (AGE(K) .GT. 15) AGE(K)=16
I=(AGE(K) + 1)/5+1
IF (W(K) .LT. 20) W(K)=20
IF (W(K) .GT. 60) W(K)=60
J=(W(K)/10) + 1
A(I,J)=A(I,J)+1
A(N,J)=A(N,J)+1
A(I,M)=A(I,M)+1
10 A(N,M)=A(N,M)+1
DO 15 I=1,N
15 WRITE(5,101) (A(I,J), J=1,M)
STOP
END

```

READ THIS ONLY

1. DO NOT EXAMINE THE OTHER PAGES OF THIS HANDOUT UNTIL INSTRUCTED TO DO SO.
2. THIS IS AN EXPERIMENT ONLY. A SERIOUS EFFORT IS EXPECTED. YOUR PERFORMANCE WILL NOT AFFECT YOUR GRADE WHATSOEVER.
3. YOUR PARTICIPATION IN THIS EXPERIMENTS IS VOLUNTARY. YOU ARE FREE TO WITHDRAW AT ANY TIME.

INSTRUCTIONS

On the next two pages is a program and its output preceded by its documentation. The documentation includes a description of the program and the correct output for set of input data. The program contains a single error. So the output following the program listing is incorrect for the input data.

Your task is to find the error in the program. When you think you have found the error, raise your hand. A proctor will tell you whether or not you have discovered the error. If you have discovered the error, then correct the error on the program listing. If not, you should continue to look for the error.

You have 20 minutes to complete this part of the experiment.

DOCUMENTATION

Purpose of program:

To calculate the total, average, minimum and maximum values of a set of observations.

Data:

NO is number of observations

NV is number of variables per observation

A(1),...,A(NO) first variable in all observations

A(NO+1),...,A(2*NO) second variable in all observations

A(2*NO+1),...,A(3*NO) third variable in all observations

etc

S(1),...,S(NO) input vector indicating which observations are to be considered in the calculations. Only observations with a non zero B(J) value will be considered.

TOTAL(1),...,TOTAL(NV) hold totals for each variable

AVER(1),...,AVER(NV) hold averages for each variable

VMAX(1),...,VMAX(NV) hold maximums for each variable

VMIN(1),...,VMIN(VN) hold minimums for each variable

INPUT DATA (NO=5,NV=3)

4.06.07.54.56.58.05.07.08.55.57.59.59.59.08.09.5
0102000405

CORRECT OUTPUT

VARIABLE	TOTAL	AVERAGE	VMIN	VMAX
1	21.0	5.3	4.0	6.5
2	27.0	6.8	5.0	8.5
3	34.0	8.6	7.5	9.5

```

PROGRAM STAT(INPUT,OUTPUT,TAPE1=OUTPUT)
INTEGER S,SCNT
DIMENSION A(15),S(5),TOTAL(3),AVER(3),VMIN(3),VMAX(3)
READ(*,100) (A(I),I=1,15)
READ(*,101) (S(I),I=1,5)
100 FORMAT(15F3;1)
101 FORMAT(5I2)
WRITE(1,101) (S(I),I=1,5)
WRITE(1,102) (A(I),I=1,15)
102 FORMAT(15F4.1)
NV=3
ND=5
DO 1 K=1,NV
TOTAL(K)=0.0
AVER(K)=0.0
VMIN(K)=1.0E10
1 VMAX(K)=- 1.0E10
SCNT=0
DO 7 J=1,NO
IJ=J-NO
IF(S(J) .EQ. 0) GO TO 5
SCNT=SCNT+1
5 DO 6 I=1,NV
IJ=IJ+NO
TOTAL(I)=TOTAL(I)+A(IJ)
IF(A(IJ) .LT. VMIN(I))VMIX(I)=A(IJ)
IF(A(IJ) .GT. VMAX(I))VMIX(I)=A(IJ)
6 CONTINUE
7 CONTINUE
IF(SCNT .EQ. 0) GO TO 15
DO 10 I=1,NV
10 AVER(I)=TOTAL(I)/SCNT
15 DO 20 I=1,3
20 WRITE(1,103) TOTAL(I),AVER(I),VMIN(I),VMAX(I)
STOP
END

```

```

1 2 0 4 5
4.0 6.0 7.5 4.5 6.5 8.0 5.0 7.0 8.5 5.5 7.5 9.5 6.0 8.0 9.5
28.5 7.1 4.0 7.7
34.0 8.5 5.0 8.5
20.5 10.1 6.0 9.5

```