OREGON STATE UNIVERSITY

COMPUTER SCIENCE DEPARTMENT

A Preliminary Investigation of the Effects of FORTRAN Data Declaration and Type Conversion Features on Program Comprehension

Murthi Nanja and Curtis Cook
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

86-60-5

A PRELIMINARY INVESTIGATION OF THE EFFECTS OF FORTRAN DATA DECLARATION AND TYPE CONVERSION FEATURES ON PROGRAM COMPREHENSION

Murthi Nanja

Curtis Cook

Computer Science Department

Oregon State University

Corvallis, OR-97331.

ABSTRACT

FORTRAN permits both explicit and implicit options for both data declaration and type conversion features. This study investigated the effects of explicit and implicit data declaration and type conversion features on a program comprehension task performed by FORTRAN programmers in an introductory programming course. The results indicated that both factors had significant effects on programmer performance and there was a significant interaction between data declaration and type conversion. In addition, hand-simulation tasks performed by the subjects provided insights on low-level errors made by the novices and misconceptions in the minds of novices.

1. INTRODUCTION

Instructors teaching novices to program face the following problems:

- What difficulties do novices experience in learning to program?
- What are the sources of these difficulties?
- What features make a language suitable or unsuitable for novice programmers?

There are no easy answers for these problems. This paper addresses some of these issues through a preliminary investigation of novices' experience while learning FORTRAN.

Studying programming errors is necessary in order to learn their causes and to devise preventive measures. A number of experiments have been performed to collect and analyze various kinds of programming errors. For example, Youngs [9] studied the errors made in programming by observing programmers working in high-level languages. Thirty beginning programmers and 12 advanced programmers each coded one or two of nine problems. Youngs required the participants to submit run logs and all the computer output from the problem. He studied programming errors in terms of error frequency, error proneness, and error diagnosis and devised methods of recording and analyzing errors. The interesting results of his research were: (a) eight language constructs accounted for 75% of all errors, with assignment and I/O accounting for 29% and 14%, respectively. (b) 75% of the errors made by beginners were semantic and logical errors. Boies and Gould [1] studied syntactic errors in FORTRAN programming and concluded that syntactic errors were not a major bottleneck in programming because they are detected by the compiler and are easy to correct.

Moulton and Muller [8] studied errors found in three months of usage of a diagnostic FORTRAN compiler by several hundred students in a introductory FORTRAN programming course. The major conclusions of their study were: (a) syntactic errors occurred in 36% of the programs analyzed by the compiler (b) arithmetic assignment statement comprised 26% of total number of compiler errors (c) arithmetic assignment statement accounted for 45.6% of all statements in a total of 5158 programs. One major static analysis of FORTRAN programs by Knuth [7] looked

at syntactically correct programs, and compiled statistics about frequency of usage of different constructs. He found that arithmetic assignment statement accounted for 51% of all statements in a sample of 440 professional programs.

Type declaration and type conversion are two programming language features common to many conventional programming languages but appearing in different forms. Fo example a variable may be statically typed (assigned in a declaration), dynamically typed (assigned during program execution), or typeless (assume type of the last value assigned to it). Two variants of static typing are implicit (done automatically) and explicit (done by programmer) typing. FORTRAN allows both implicit and explicit typing of REAL and INTEGER variables. In FORTRAN a variable may be explicitly assigned either type in a declaration or implicitly assigned a type according to the first letter of the variable name.

Type conversion may be explicit or implicit. In explicit type conversion the programmer must specify all type conversions while in implicit type conversion the type conversion is performed automatically as needed such as when the variable is an operand in an expression. As a sort of middle ground FORTRAN gives the programmer the option of explicitly converting a numeric operand or letting an implicit conversion occur.

There is no common agreement as to which is the best method of type declaration or type conversion. In one study Gannon [5] found that far fewer errors were made in a statically typed language than a typeless language. However, there is considerable debate about which is best for FORTRAN because it permits both explicit and implicit data declarations and type conversions for REAL and INTEGER variables.

Arguments in favor of implicit data declaration are programmers can remember the types of variable names through clues in the name or comments and it is convenient and simpler to use the implicit declaration convention throughout the program.

Many programmers favor implicit type conversions because the conversion rules are simple and do not occur that often.

Some arguments supporting explicit typing of variables are:

- FORTRAN rules for determining type from the initial letter of the variable name do not protect against misspelling and restricts the choice of integer variable names.
- Programmer has to use type statement for all the FORTRAN data types other than INTEGERs and REALs.
- Explicitly naming the type ensures that the programmer is conscious of each variable name, and its type.

Similarly, some arguments favoring explicit type conversion feature over implicit type conversion feature are as follows:

- Programmer can minimize his or her "mental load" when understanding FORTRAN expressions.
- Explicit type conversion ensures that the programmer is conscious of the data type of each variable name.

Further, many FORTRAN textbooks advocate explicit type declaration and explicit type conversion for obtaining better quality programs. For example:

> "While the truncation resulting from integer or the assignment of a real value to an integer variable can be useful in programming problems, it can have disastrous effects when it occurs by accident (because of programmer error). We therefore suggest that you always explicitly specify the conversions involved in a mixed expression or assignment (integer to real or real to integer)."
>
> ----Friedman, F.L., and Koffman, E.B. [4, p.143]

> "If you have mixed-mode operations for operations other than exponentiation, use the functions INT and REAL so that operations use all integers or real values."

> "Explicit typing is an aid in program clarity as well as in error detection. Some programmers prefer to list all variables on specification statements, including those correctly typed by the implicit rules."

----Etter, D.M. [2, p.64-65]

We considered the following two programming factors for our preliminary investigation: FORTRAN type declaration and type conversion. We selected these factors for two reasons. First, there seem to be no studies that either to justify or to reject the explicit type declaration and conversion recommendations proposed in many FORTRAN textbooks. Therefore we were curious as to whether the use of explicit type declaration and explicit type conversion aids program comprehension. Second, since many studies identified FORTRAN arithmetic assignment statement as a major source of programming errors, we investigated the kinds of arithmetic operator and library function errors commonly made in assignment statements.

This paper presents the results of a controlled experiment that compared the effects of FORTRAN type declaration and type conversion features on a program comprehension task performed by novice programmers. In addition, we collected the low-level errors made by novices in comprehending FORTRAN assignment statements. The subjects were presented one FORTRAN program and were instructed to hand-simulate the program by drawing a tree-like structure form for each executable statement. The tree-like structures illustrated the order in which subjects hand-executed the arithmetic operators and showed the intermediate values resulting from partial evaluation of all assignment statements. The tree-like structure provided a protocol analysis of the step-by-step hand execution of each statement. Our results suggest that programs with implicit type declaration and implicit type conversion are less comprehensible to inexperienced FORTRAN programmers. Also most low-level errors involved mixed mode division and the MOD library function.

2. THE EXPERIMENT

The study had two primary goals:

1. To examine the effects of FORTRAN implicit and explicit type declaration and type conversion features on program comprehension by novice programmers.
2. To identify common errors made by novice FORTRAN programmers in understanding

FORTRAN arithmetic operators and common library functions.

2.1 Subjects

Eighty four students enrolled in an introductory FORTRAN programming course at Oregon State University served as subjects in the experiment. Most of the subjects had previous experience with one or more of other programming languages such as BASIC and Pascal. The subjects were randomly divided into four groups each consisting of twenty one subjects. The subjects' background information is shown in Table 1. A series of chi-square tests on the background information variables showed no statistically significant differences among the four groups after the random assignment of subjects. Hence the random assignment was successful in controlling for programmer performance such as experience and knowledge of different languages, etc. All subjects were unpaid volunteers.

2.2 Design and Material

The experiment was designed to study the effects of two independent variables: the type declaration, and the type conversion. A 2 by 2 (Type Declaration by Type Conversion) factorial design was employed. Each factor had two levels of treatment: explicit type declaration (ETD) , and implicit type declaration (ITD) ; explicit type conversion (ETC), and implicit type conversion (ITC). There were two dependent variables: accuracy of correct response for each statement, and number of low-level errors.

Four FORTRAN programs that implementing the same algorithm [6] for determining the date of Easter Sunday for any year after 1582 were constructed. Two levels of type declaration and type conversion were defined for each program. Programs were between 27 to 29 lines in length, and consisted of 18 arithmetic assignment statements, two logical IF statements, and one IF-THEN-ELSE construct. Two programs had explicit type declaration and two did not. No iteration constructs were included in the programs. Four to five blank lines were provided below each statement in the program so that subjects could draw the tree-like structure that show their order of mental execution of that statement. No meaningful variable names were used in any of

the programs. Two sample programs along with instructions given are included in Appendix A. In order to save space, the 4-5 blank lines inserted between program statements are shown only for the first program listing in the Appendix. These four programs were functionally equivalent, but they differed in nontrivial ways due to choices of type declaration and type conversion features.

2.3 Procedure

The experiment was conducted as a 50 minute quiz three weeks prior to the spring term final examination. The instructor of the course had described the purpose of the quiz in the previous lecture. Subjects were randomly assigned to each of the four experimental conditions. Programs were presented to the subjects in random order so that order would be balanced. The subjects were first asked to provide their background information about computer programming languages and then instructed to hand-simulate the program given on the following page of the material by drawing tree-like structure in the space below each statement. As an example, a tree-like structure is shown in Figure 1. The tree-like structures had been used extensively in regular lectures to illustrate the evaluation of expressions. Although subjects were given 50 minutes to perform the comprehension task, almost all of them finished in about 30 minutes.

3. RESULTS

Each statement in the program was graded on a statement by statement basis to compensate for error propagation from one statement to another statement within the program. Twenty-three statements were graded. Subjects made the following operation and conversion errors: (a) misunderstood the function of MOD operator, (b) reversed the argument order for MOD, (c) executed real division instead of integer division, (d) applied wrong arithmetic operation, and (e) did not convert data types in a mixed mode assignment statement. A subject received one point only if all of the low-level operations and conversions in the statement were performed correctly. Otherwise the subject received a score of zero.

Table 2 presents the means and standard deviations for correct response score. An analysis

of variance was performed with type declaration and type conversion as the independent variables and the correct response score and the number of errors for each of the four programs as the dependent variables. The results indicate that explicit type declaration and explicit type conversion groups supplied significantly more correct responses. Specifically, effects of both type declaration and type conversion were statistically significant for correct response score, $F(1,80) = 8.74, 4.59$, $p < 0.01$, $p < 0.05$. A similar result was obtained for errors committed by subjects. i.e. $F(1,80) = 10.13, 15.0$, $p < 0.01$. However, the interaction effect was significant only for number of errors, $F(1,80) = 3.75$, $p < 0.05$.

The mean response score, and the mean error are plotted as functions of type declaration and type conversion in Figure 2. It is evident from the curves that explicit type conversion (a) improves program comprehension score, and (b) minimizes the number of procedural errors.

In addition to analysis of variance, a multiple comparison test using the studentized range (Newman-Keuls method) [3] was conducted on the mean error data. The results indicated that, the differences between ITD-ITC and ETD-ETC, between ITD-ITC and ETD-ITC, and between ITD-ITC and ITD-ETC were significant, $Q = 5.97, 4.29, 3.79$, $p < 0.05$. No other comparisons were significant.

Table 3 displays the percentage of subjects in each group who gave the correct solution without making any error. Surprisingly, these figures are quite low.

From the tree-like structure, it was possible to gather further information about particular types of low-level mistakes novices made in hand-executing the program. We identified five categories of low-level errors and classified all errors into one of the following categories: mode error, assignment operator error, library function error, computation error, and miscellaneous error. Mode error occurred when a real expression was evaluated in integer mode or was evaluated in real mode and an assignment operator error resulted from omitting type conversion across the equal sign of an arithmetic assignment statement. A library function error resulted from misunderstanding the function or incorrectly using its arguments. misuse of its arguments and misunderstanding of its function. Computation error resulted from programmers' carelessness or

ignorance in manipulating arithmetic operators. Miscellaneous errors included all other types of errors such as clerical error, illegal application of operator error, operator precedence error, and other unknown errors.

Table 4 shows the total number of errors and the proportion of errors in each error category. As can be seen from this table, nearly one-half (0.42) the error occurred in implicit data declaration and implicit type conversion group. As might be expected over 95% of all assignment operator errors occurred in the group with implicit type conversion and nearly two-third of the mode and library function errors occurred in the group with implicit data type. The chi-squared test revealed that four groups differed significantly with respect to the proportion of errors made in each error category. i.e. $X^2 = 52.27$, $p < 0.001$.

Table 5 shows the distribution of errors by statement for all versions of the program. The table also displays the total number of errors that occurred in each program version as well as in each statement across all programs. As can be seen from the table, a few program statements contain a large percentage of the errors. For example, five statements had more than 30 errors, three had more than 10 errors, and the remaining statements had less than 10 errors. A closer examination of these high error frequency statements reveals that they all use either MOD operator or integer division operator. Moreover, most of these error occurred in program with the implicit type conversion. We also observed that novices made these same errors at several places in the program.

4. MISCONCEPTIONS AND RECOMMENDATIONS

Studying inexperienced programmers' errors provides insight about errors they make and misconceptions they have. The error categories reported in this paper reflect the novice programmers' confusion and lack of low-level knowledge about the FORTRAN division operator and library functions. Following is a description of novices' misconceptions associated with three important error categories in FORTRAN and possible hypotheses which could account for these misconceptions.

4.1 Mode Error

A mixed-mode expression in FORTRAN contains operands with two or more data types. Mode error results from incorrect evaluation of a mixed-mode arithmetic expression. The mode error may be due to programmers' long time association with real arithmetic operations in mathematics. Novices treat an integer expression as real value most of the time. Since most of the mode errors involved integer division a possible explanation is the confusion over division operator (/) because FORTRAN uses this same operator symbol for both integer and real division. Other programming languages such as Pascal and C carefully avoid this problem by using two different operator symbols, one for integer division and the other for real division. This appears to underscore the difference between real and integer division and remind novices of the difference. Furthermore, novices are not aware of the conversion rules that apply to various arithmetic operators. Only a few FORTRAN text books explicitly show all conversion rules for arithmetic operators. Since most novices' mode errors are due to not understanding the type of mixed mode results, we strongly feel that the instructors teaching FORTRAN should place more emphasize on type conversion rules for real and integer and in particular expression involving integer division operator. For example a simple rule such as the type of the result is integer only if both operands are of type integer. Otherwise the result is real.

4.2 Assignment Operator Error

FORTRAN allows type conversion across the equal sign. Such conversion is implicit in FORTRAN; omitting such conversion generates assignment operator error. It is interesting to note that novices perceive FORTRAN statements such as IR=XY and IR=IXY where XY=191.0 and IXY=191 as equivalent statements. This probably shows novices lack knowledge of the internal data representation. It may also reflect a confusion about the equal sign in traditional algebra with the assignment operator in programming. One possible way to avoid this error is to teach novices how to use explicit type conversion functions (INT and REAL) in assignment statements and to emphasize different representation for REAL and INTEGER.

4.3 Library Function Error

Every library function in FORTRAN takes a certain number of arguments in a particular order; misusing these arguments creates library function error. Our data indicated that a few novices interpreted the first argument of a MOD function as divisor and the second argument as dividend, whereas many others misunderstood the function of MOD function. The unfamiliarity of this library function seemed to account for majority of these errors.

5. CONCLUDING REMARKS

This paper has presented an initial study of explicit and implicit options associated with type declaration and type conversion features in FORTRAN and analyzed the low-level errors made by inexperienced FORTRAN programmers. In summary we believe that our experimental results suggest the following:

- When type declaration and type conversion features of FORTRAN are specified explicitly in a program, they improve novice programmers' comprehension.
- When programs incorporate implicit options for type declaration and type conversion features, novices tend to make more errors in comprehending the program.
- Mode error, assignment operator error, and library function error represent novices' large percentage of arithmetic operator errors in FORTRAN.

However, our results cannot be generalized for the following reason: the task used in our study required only one small program and the subjects were not professional programmers.

The empirical evidence gathered in this research helped in identifying programming difficulties of novices in understanding FORTRAN arithmetic operators and their sources. Based on the qualitative analysis of the magnitude and kind of errors we make the following teaching recommendations for beginning students:

- Encourage novices to write programs with explicit options for data typing of variables and type conversion in arithmetic assignment statements.

- Give more emphasize to mixed mode arithmetic expressions, especially integer division operator.
- Provide complete simple to remember conversion rule tables for all arithmetic operators.
- Cover a few of common library functions such as MOD throughly.

REFERENCES

[1] Boies, S.J., and Gould, J.D. Syntactic errors in computer programming. *Human Factors*, 16, pp. 253-257.

[2] Etter, D.M. *Problem solving with structured FORTRAN 77.* The Benjamin/Cummings Publishing Company, Inc.. 1984.

[3] Ferguson, G.A. *Statistical Analysis in Psychology & Education.* McGraw-Hill Book Company, NewYork, 1971.

[4] Friedman, F.L., and Koffman, E.B. *Problem solving and structured programming in FORTRAN.* Addison-Wesley Publishing Company, 1981.

[5] Gannon, J.D. An experimental evaluation of data type conventions. *Communications of the ACM,* 20, August 1977, pp. 584-595.

[6] Knuth, D.E. *Fundamental algorithms - The art of computer programming.* Vol. 1, Addsion-Wesley Publishing Company, 1973.

[7] Knuth, D.E. An empirical study of FORTRAN programs. *Software - Practice and Experience,* Vol. 1, No. 2, 1971, pp. 105-133.

[8] Moulton, P.G., and Muller, M.E. DITRAN -- a compiler emphasizing diagnostics. *Communications of the ACM,* 10, January 1967, pp. 45-52.

[9] Youngs, E.A. Human errors in programming. *Int. J. Man-Machine Studies.* Vol. 6, No. 3, May 1974, pp. 361-376.

Table 1
Subjects' background information

Background information	Group 1	Group 2	Group 3	Group 4
Major				
Computer science	4	3	4	2
Engineering	13	16	13	16
Others	4	2	4	3
Year in school				
Freshman	12	11	10	11
Sophomore	4	6	4	4
Junior	1	2	2	2
Senior	3	2	3	3
Others	1	0	2	1
Programming languages				
Pascal	16	17	15	13
Basic	12	15	14	14
Fortran	11	11	9	12
Others	6	3	4	5

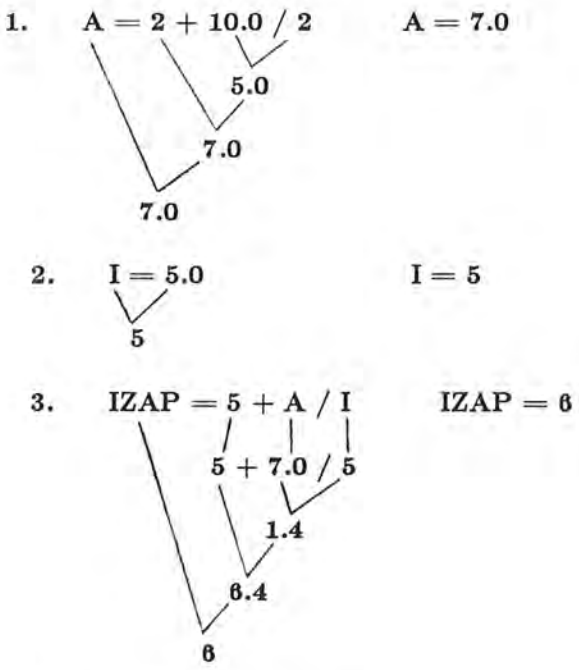


Figure 1. Tree-like structure

Table 2
Mean and standard deviation for four versions of the program

	ETD	ITD
ETC		
Overall response score		
Mean	20.66	19.40
Std dev	2.37	2.48
N	21	21
Number of errors		
Mean	2.33	3.05
Std dev	2.46	2.42
N	21	21
ITC		
Overall response score		
Mean	19.90	17.38
Std dev	2.93	2.85
N	21	21
Number of errors		
Mean	3.43	6.24
Std dev	2.58	2.43
N	21	21

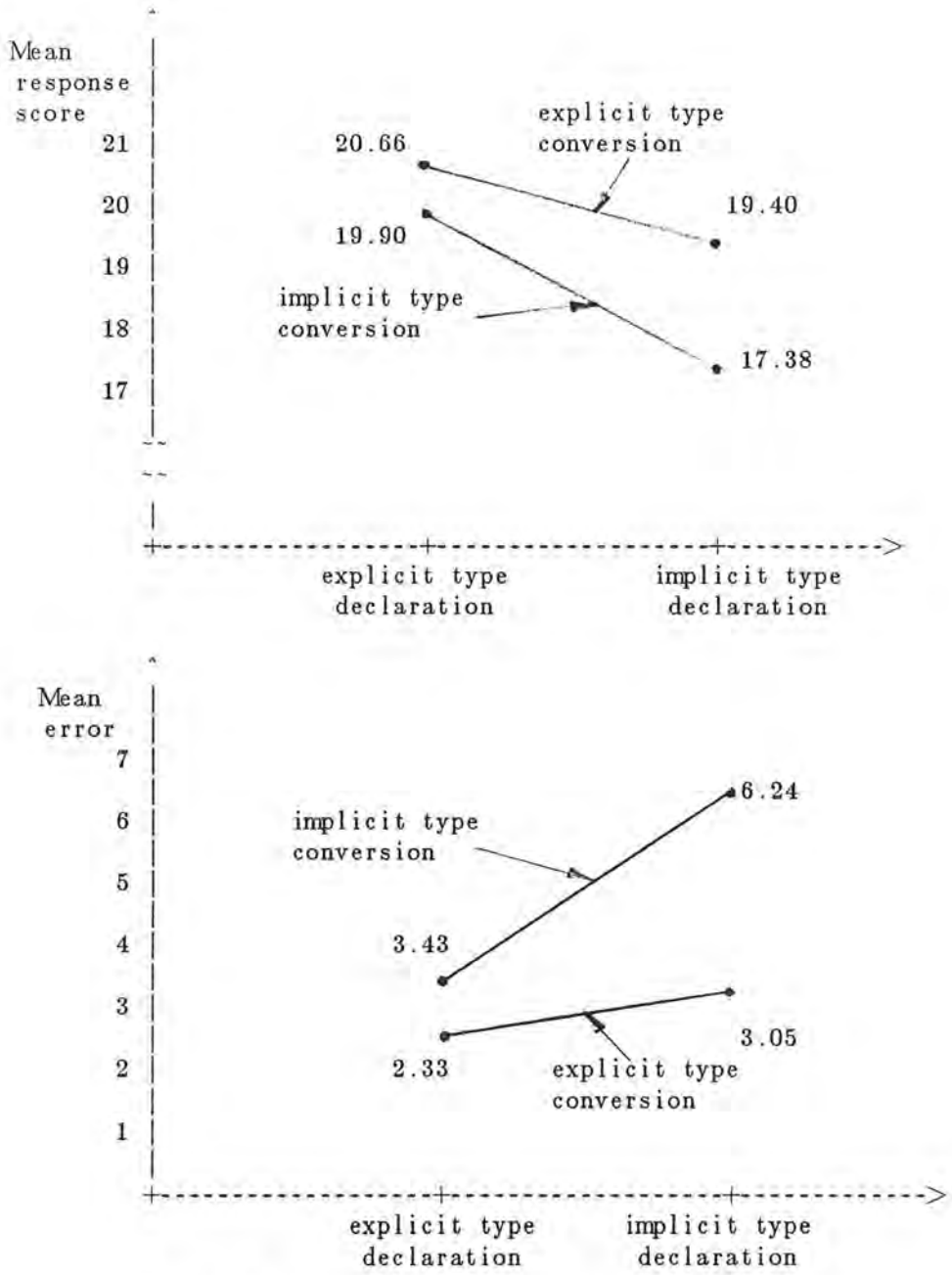


Figure 2. Type declaration and type conversion
(a) mean response score (b) mean error

Table 3
Overall performance

Group	Number of correct solutions
ETD-ETC	38 % (8/21)
ETD-ITC	33 % (7/21)
ITD-ETC	14.3 % (3/21)
ITD-ITC	9.5 % (2/21)

Table 4
Classification of low-level errors

Group	Type of errors						
	AOE	MDE	LFE	CPE	MSE	Total	Proportion
ETD-ETC	1	13	13	12	10	49	0.15
ETD-ITC	24	17	14	14	3	72	0.23
ITD-ETC	6	27	28	11	7	79	0.20
ITD-ITC	41	22	25	22	22	132	0.42
Total	72	79	80	59	42	332	1.00
Proportion	0.21	0.22	0.24	0.19	0.13	1.00	

Legend:
AOE ----> assignment operator error
MDE ----> mode error
LFE ----> library function error
CPE ----> computation error
MSE ----> misellaneous error

Table 5
Error distribution by statement

Statement number	ETD-ETC	ETD-ITC	ITD-ITC	ITD-ETC	Total
1					
2					
3					
4		7	12		19
5	9	8	13	12	42
6	8	8	15	13	44
7		6	3		9
8	1		4	1	6
9	3	2	8	5	18
10	2	4	3	1	10
11	8	10	15	12	45
12			4	1	5
13	3	6	7	2	18
14		2	2	1	5
15	6	9	16	14	45
16			3		3
17	1		4	2	7
18		2	2		4
19		1	3	2	6
20			3		3
21	7	7	12	13	39
22	1		3		4
23					
Total	49	72	132	79	332

APPENDIX A

NONCREDIT-QUIZ
Instructions

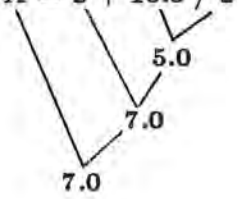
Read the following instructions carefully:

1. This quiz does not affect your course grade in any way.
2. If you have questions during the test, please raise your hand.
3. You are not permitted to refer to any textbooks and/or lecture notes during the test.
4. Please provide the information requested below :

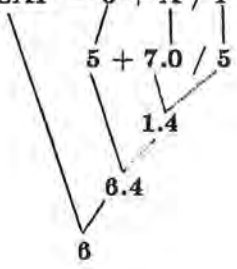
   Major :
   Year in School :
   Programming Languages Known :

5. On the following page you are given a FORTRAN program. Show the values of all variables after this program is executed. You must explicitly show your mental calculations for each statement in the program as shown in the following examples. You can assume implicit typing for undeclared variables. When you are done, write down the approximate time taken to complete this task.

1. **A = 2 + 10.0 / 2** **A = 7.0**



2. **IZAP = 5 + A / I** **IZAP = 6**



Program for ETD-ETC group

Line #				
1	PROGRAM KNUTH(INPUT,OUTPUT)			
2	INTEGER GA,XB,AC,YD,DE,MN,DS,IT,NU,CE,IA,JF,NY			
3	REAL NX,CF,IZ,JE			
4	YD = 1991	YD	=	_____
5	GA = MOD(YD,19) + 1	GA	=	_____
6	CF = REAL(YD/100+1)	CF	=	_____
7	CE = INT(CF)	CE	=	_____
8	MN = INT(3.0*CF)	MN	=	_____
9	XB = MN/4-12	XB	=	_____
10	DS = 8*CE+5	DS	=	_____
11	IZ = REAL(DS/25-5)	IZ	=	_____
12	IA = INT(IZ)	IA	=	_____
13	DE = 5*YD/4-XB-10	DE	=	_____
14	IT = 11*GA+20+IA-XB	IT	=	_____

```
15    JE = REAL(MOD(IT,30))                                  JE  =  _____

16    JF = INT(JE)                                           JF  =  _____

17    IF (JE.EQ.25.0.AND.GA.GT.11.OR.JE.EQ.24.0)JF=JF+1      JF  =  _____

18    NX = REAL(44-JF)                                       NX  =  _____

19    IF (NX.LT.21.0)NX=NX+30.0                              NX  =  _____

20    NY = INT(NX)                                           NY  =  _____

21    NU = NY+7-MOD(DE+NY,7)                                 NU  =  _____

22    IF (NU.GT.31) THEN

23       PRINT *,JF,NX                                       JF  =  _____
                                                             NX  =  _____
24       PRINT *,(NU-31),'APRIL',YD
         ____________________________
25    ELSE
26       PRINT *,JF,NX                                       JF  =  _____
                                                             NX  =  _____
27       PRINT *,NU,'MARCH',YD
         ____________________________
28    ENDIF
29    END
```

Program for ITD-ITC group

Line #				
1	PROGRAM KNUTH(INPUT,OUTPUT)			
2				
3				
4	IY = 1991.0	IY	=	_____
5	IG = MOD(IY,19) + 1.0	IG	=	_____
6	CF = IY/100+1.0	CF	=	_____
7	IC = CF	IC	=	_____
8	MN = 3*CF	MN	=	_____
9	KX = MN/4-12.0	KX	=	_____
10	LD = 8.0*IC+5	LD	=	_____
11	ZI = LD/25-5.0	ZI	=	_____
12	IZ = ZI	IZ	=	_____
13	ID = 5*IY/4-KX-10.0	ID	=	_____
14	IT = 11.0*IG+20.0+IZ-KX	IT	=	_____
15	EJ = MOD(IT,30)	EJ	=	_____
16	JE = EJ	JE	=	_____
17	IF (EJ.EQ.25.AND.IG.GT.11.0.OR.EJ.EQ.24.0)JE=JE+1.0	JE	=	_____
18	XN = 44.0-JE	XN	=	_____
19	IF (XN.LT.21.0)XN=XN+30	XN	=	_____
20	NX = XN	NX	=	_____
21	NU = NX+7.0-MOD(ID+NX,7)	NU	=	_____
22	IF (NU.GT.31) THEN			
23	PRINT *,JE,XN	JE	=	_____
		XN	=	_____
24	PRINT *,(NU-31),'APRIL',IY			
25	ELSE			
26	PRINT *,JE,XN	JE	=	_____
		XN	=	_____
27	PRINT *,NU,'MARCH',IY			
28	ENDIF			
29	END			