UC Irvine
ICS Technical Reports

Title
A System for Testing Student Programs

Permalink
https://escholarship.org/uc/item/9fk0v3sg

Author
Tonge, Fred M.

Publication Date
1971-07-01

Peer reviewed

eScholarship.org Powered by the California Digital Library
University of California

12

A SYSTEM FOR TESTING STUDENT PROGRAMS

Fred M. Tonge

TECHNICAL REPORT No. 12, JULY 1971

INTRODUCTION

TEST is a system for checking student programs using data sets supplied by the instructor. This system owes much to the TEACH system developed at Dartmouth for grading student BASIC programs. However, the goals of TEST are somewhat different.

1. To as large an extent as possible, the system is language independent. This is achieved primarily by describing the language of student programs in production form, so that by changing only the productions, student programs in different languages can be handled.
2. Testing programs are written in the same language as student programs.
3. The system is used to test student programs, not to grade them. No data on errors, number of attempts, is kept.
4. Following from 3., there is no need that the system be "safe" from students; indeed, students can be encouraged to figure out how TEST works.
5. Following also from 3., the system can assume legal student programs; no large effort need be put into detecting and/or avoiding tricks.

The current system works with XBASIC on the Sigma-7, and (although it need not be) TEST is programmed in XBASIC. It has not yet been used with another language, but that does seem straightforward. A description of the TEST program itself is given in Appendix B.

HOW TEST APPEARS TO THE STUDENT

The student who is going to use TEST is given both a general description of how to prepare a program for TESTing (Figure 1) and a problem assignment (Figure 2). When the student has a (hopefully) debugged version of the program filed (e.g., Figure 3), he uses TEST, as shown in the sample output (Figure 4).

HOW TEST APPEARS TO THE INSTRUCTOR

The instructor first prepares the problem assignment. Then, based on instructions for TESTing a program (Figure 5), he prepares a program that will be merged with the student's program to test that program. (Figure 6 gives a flowchart of a testing program for the problem of Figure 2, and Figure 7 gives a program for that flowchart.) When the testing program is debugged, it is filed under the appropriate name in the TEST account, and the system is ready for student use.

HOW TEST WORKS

The TEST system proceeds in the following manner:

1. The file designated by the student is read in and edited, and a new file (COPYFILE) is produced containing the edited program.
2. The COPYFILE is loaded.
3. The test program is loaded, merging with COPYFILE.
4. The combined program is run.

The exact details of merging two programs and of executing a program depend on the language and operating system in which student and test programs are run; in XBASIC under BTM this is straightforward.

The editing done by TEST is controlled by a description of the syntax of the language being processed. For XBASIC, the changes made in an edited program are as follows. (Most of the complications in handling XBASIC arise from the possibility of compound conditional statements.)

a. Line numbers are checked, and for any less than 10 or greater than 9999 an error indication is printed.

b. The command words 'INPUT' and 'INPUTS' are replaced by 'READ'.

c. Statements beginning with 'STOP', 'PAUSE', or 'END' are replaced by the statement 'GOTO 10000' (the beginning of the test program).

d. Statements beginning with 'PRINT' are replaced by the statement '0=0" (an effective no-operation).

The edited version of TRI (e.g., COPYFILE) is shown in Figure 8.

HOW TO CHANGE LANGUAGES

Changing the TEST system to handle some other language for student and instructor programs than XBASIC requires several changes in the present system, and also the appropriate file linking capability in the processor for the new language.

1. The productions controlling editing of the student program must be changed to those appropriate for the new language. The productions for XBASIC are described in Appendix C.

2. The scanner in TEST may require modification to recognize the basic classes of tokens for the new language.

3. The statement in the TEST system which outputs to the edited

program (COPYFILE) a linkage command to the test program (in the current version of TEST, statement 5000) must be changed to reflect the linkage conventions of the new language.

4. The commands at the end of TEST to link it to COPYFILE must be changed to reflect the linkage conventions of the new language and the fact that TEST and COPYFILE are in different languages.

5. The commands added to the end of the test program to cause its execution after linking with the edited program (see Figure 5) must be changed to reflect the conventions of the new language.

Preparing a Program for TESTing

Some of your programming assignments can be tested using the system TEST. For those assignments, when you believe your program is debugged and ready to be tested, file it under the name given in the assignment and then proceed as follows. (Suppose the specified file name is TRI.)

```
!GET TEST
FILE NAME: TRI          (You type the underlined parts.)
```

TEST will load your program from the file, modify it slightly for TESTing, and then try it out with several sets of data, informing you of any errors it finds. If there are errors, you can correct them, file the revised program, and TEST it again.

TEST expects that the program you give to it is a working one; it may not accept an undebugged program. In preparing a program for TEST, you must follow these rules. If you do not, TEST may give you inaccurate results.

1. Statement numbers must lie between 10 and 9999.
2. Variable names starting with the letter O followed by one or more digits should not be used.
3. The program should input (whether by INPUT or READ) only that data described in the problem statement. Other input statements will confuse TEST.
4. The program should do exactly what is specified. For example, if the problem calls for processing one set of data, the program should not loop back for more sets. If it does, TEST may give inaccurate results.
5. TEST cannot stop a program that loops forever. You will have to interrupt if that is the case.

TEST is not a grading program. You are invited (encouraged) to find out how it works. And it may still contain a few bugs, too. So if you can't explain its treatment of your program, let us know.

Figure 1.

Triangle Program Assignment

Write a program to read in three numbers which are the lengths of sides of a triangle. Compute the type of triangle they form, if any, and set the variable F according to that result as follows.

F = 0 if not a triangle
F = 1 if an acute triangle
F = 2 if a right triangle
F = 3 if an obtuse triangle.

If you wish to use TEST to check your program, file it under the name TRI and then GET TEST.

Good luck.

Figure 2.

```
15          REM**TRIANGLE ASSIGNMENT
20 READ A,B,C
30 X=MAX(A,B,C)
35 Y=A+B+C-2*X
37 IF Y <=0 THEN 120
40 Y=X 2
50 Z=A 2+B 2+C 2-2*Y
60 IF Z<0 THEN 100
70 IF Z>0 THEN 110
80 F=2
90 GO TO 115
100 F=1
105 GOTO 115
110 F=3
115 PRINT A,B,C,F
116 IF F=0 THEN PRINT 'NOT TRIANGLE' ELSE PRINT 'TRIANGLE'
117 STOP
120 F=0
125 GOTO 115
150 DATA 1,2,4
160 DATA 4,5,6
170 DATA 3,4,5
180 DATA 2,3,4
190 DATA -2,1,2
200 END
```

Figure 3

```
!GET TEST
FILE NAME:  TRI

BEGIN TESTING TRI

OKAY FOR  1    2   4

ERROR
FOR  4   5   6 PROGRAM SET F= 3, SHOULD BE  1

OKAY FOR  3    4   5

ERROR
FOR  2   3   4 PROGRAM SET F= 1, SHOULD BE  3

OKAY FOR -2   -2  -3

TEST COMPLETED
HALT AT 10090
```

Figure 4.

Preparing a TEST Program

The program which you prepare to test a student's program will be merged with that program and the two executed together. The student program will be in lines 10 through 9999, so your program should proceed and follow that area.

In preparing a TEST program, use the following rules.

1. Use only variable names beginning with the letter O followed by one or more digits.

2. Use statements numbered 1-8 for initial setup, before first exceution of the student program.

3. Include a statement 9, a REM or other no-op, as an entry point to the student program.

4. The student program will exit to statement 10000 when completed, so checking his results should start there.

5. Make sure that your instructions to the student specify the variables that he is to set as output, whether or not his program is to loop through several data sets, any intermediate variables that you want him to use specifically, and the name under which he is to file his completed program.

When your test program is completed and debugged, file it in the test program account under the same file name as the student is to use, but with a * appended (e.g., the test program for TRI would be filed under TRI*). Then add to that file the following two lines, using the Edit subsystem:

```
RUN
X
```

Figure 5.

Procedure for Testing TRI

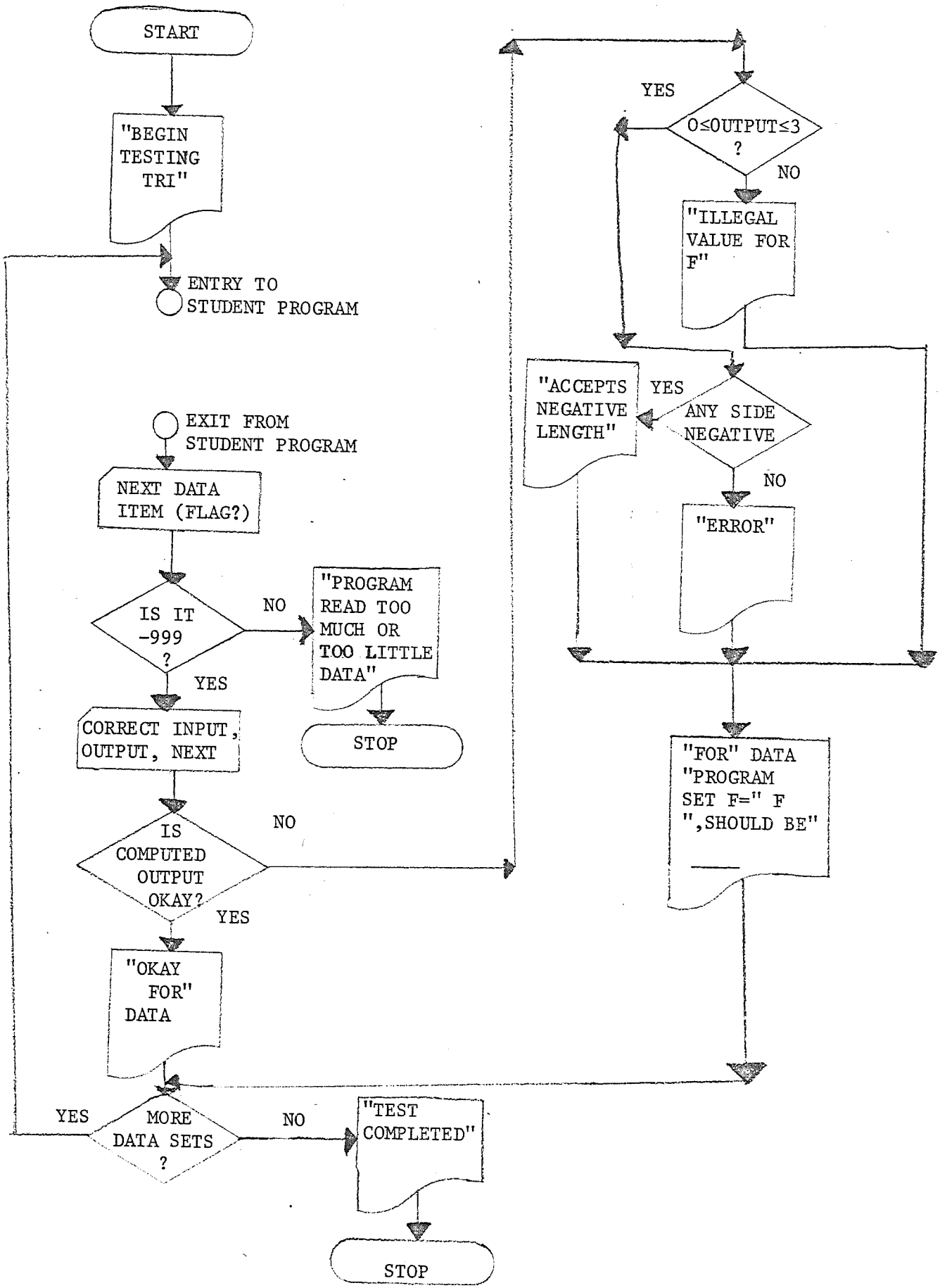

Figure 6.

TRI*

```
 1 PRINT
 2 PRINT 'BEGIN TESTING TRI'
 3 PRINT
 9 REM          ENTRY FOR STUDENT'S PROGRAM
10000 REM      ENTRY FOR TEST PROGRAM
10030 READ A2
10040 IF A2#-999 THEN 10100 ELSE READ A3,A4,A5,A6,A7
10050 IF A6#F THEN 10120 ELSE PRINT 'OKAY FOR ';A3;A4;A5
10060 PRINT
10070 IF A7=1 THEN 9 ELSE PRINT
10080 PRINT 'TEST COMPLETED'
10090 STOP
10100 PRINT  PROGRAM READ TOO MUCH OR TOO LITTLE DATA
10110 STOP
10120 IF F<0 OR F> 3 THEN 10170
10130 IF MIN(A3,A4,A5)>0 THEN PRINT 'ERROR' ELSE PRINT 'ACCEPTS NEGATIVE LENGTH
10140 PRINT 'FOR ';A3;A4;A5;' PROGRAM SET F=';F;', SHOULD BE ';A6
10150 PRINT
10160 GOTO 10070
10170 PRINT  ILLEGAL VALUE FOR F
10180 GOTO 10140
10190 DATA 1,2,4,-999,1,2,4,0,1
10200 DATA 4,5,6,-999,4,5,6,1,1
10210 DATA 3,4,5,-999,3,4,5,2,1
10220 DATA 2,3,4,-999,2,3,4,3,1
10230 DATA -2,-2,-3,-999,-2,-2,-3,0,0
10240 END
RUN
X
```

Figure 7

"COPYFILE"

```
15          REM**TRIANGLE ASSIGNMENT
20 READ A,B,C
30 X=MAX(A,B,C)
35 Y=A+B+C-2*X
37 IF Y <=0 THEN 120
40 Y=X 2
50 Z=A 2+B 2+C 2-2*Y
60 IF Z<0 THEN 100
70 IF Z>0 THEN 110
80 F=2
90 GO TO 115
100 F=1
105 GOTO 115
110 F=3
115 O=O
116 IF F=0 THEN O=O ELSE O=O
117 GOTO 10000
120 F=0
125 GOTO 115
150 REM 1,2,4
160 REM 4,5,6
170 REM 3,4,5
180 REM 2,3,4
190 REM *2,1,2
200 GOTO 10000
LOAD 'TRI*(ICTEST)'
```

Figure 8

(This is the edited version of the program in Figure 3.)

APPENDIX A
REALITIES

The preceding description of how TEST works is in fact idealistic, as of the present moment, and must be changed slightly to work under the current version of XBASIC and TEST.

First, XBASIC itself cannot be used, because it's file handling capabilities are not complete; rather, an experimental version of XBASIC called Z must be used. Assuming that the student had in his files a copy of the TEST system named XTEST, he could use it as follows:

```
!GET Z

>LOAD 'XTEST'
FILE NAME:  TRI

BEGIN TESTING TRI

OKAY FOR  1    2   4

ERROR
FOR  4   5   6 PROGRAM SET F= 3, SHOULD BE 1

OKAY FOR  3    4   5

ERROR
FOR  2   3   4 PROGRAM SET F= 1, SHOULD BE 3

OKAY FOR -2   -2  -3

TEST COMPLETED
HALT AT 10090
```

If XTEST were available in a common account (say, XTEST), then the LOAD given above could be replaced by LOAD "XTEST(XTEST)" and there would be no need for a copy of XTEST in the student's files.

Also, the current procedure leaves the edited program COPYFILE as a file in the student's account. He must be warned to delete it after testing if he does not want to be charged for the space (and he must have the space to create COPYFILE in the first place or the TESTing procedure will not work).

APPENDIX B
THE TEST PROGRAM

On the following pages are a listing of the TEST program in XBASIC. The major part of the program is an adaption of a TYMSHARE SuperBasic program and contains a number of possibilities for improved programming.

A number of the important variables used in TEST are listed below:

```
G0   line being scanned
G1   index of character in line
G2   depth of stack of scanned characters
G3   index in production picture being matched to stack
G7   index of current production
G8   index of stack of scanned characters
A    stack of scanned character types (see below)
H    stack of actually scanned tokens corresponding to A
G    array of production pictures
GS   array of replacement pictures
K    array of keywords
Q    array of subroutine entries
J    array of productions -- each row a production
       col 1: replacement picture index
       col 2: semantics routine number
       col 3: index of next production on success
                (1000=DONE; 2000=SUBROUTINE RETURN)
       col 4: scan next character indication
       col 5: index of next production on failure
                (1000=DONE; 2000=SUBROUTINE RETURN;
                 -number= error; 0=numerically next production)
F    column from which to start copying valid output
```

The input statement scanner within TEST produces both the scanned token and an indication of the type of token. The types recognized in processing XBASIC are:

```
%    name
#    number
$    string
=    operator
,    separator
```

APPENDIX B (continued)

Production pictures may match any of these or certain other cases. Each token in a production picture is encoded in two characters. If one of the above, the token is blank followed by the type symbol. Keywords are encoded by two digit numbers giving their index in the keyword array. Subroutine entries are encoded by a single digit giving the index of the entry in the subroutine array followed by the character @. An indication of "match anything" is given by the two characters ? .

```
7 READ PR,KW,RP,SB                          Set up data--productions
8 FOR J1=1 TO PR
9 READ G9
11 G(J1)=RIGHT((SPACE(10)+G9),10)             - stack pictures
13 NEXT J1
15 FOR J1=1 TO KW
16 READ K(J1)                                 - keywords
17 NEXT J1
18 FOR J1=1 TO RP
19 READ GS(J1)                                - replacement pictures
20 NEXT J1
21 FOR J1=1 TO SB
22 READ Q(J1)                                 - subroutine entries
23 NEXT J1
24 FOR J1=1 TO PR
25 FOR J2=1 TO 5
26 READ J(J1,J2)                              - production information
27 NEXT J2,J1
28 PRINT  FILE NAME:  :                     Get File name
29 INPUTS FILE
30 ENDFILE 5000
32 READS FROM FILE:G0                       Read next line from file
35 G2,G3,G6,G7=0
36 G0=G0+ @
38 G1=1
39 GOSUB 1010                               Scan 1st character
40 REM
44 G7=G7+1                                  Get next production
50 G3=10
60 G8=G2
70 G9=SUBSTR(G(G7),G3,1)                    Scan stack picture character of prod.
80 IF G9=' ' THEN 230 ELSE IF G9='.' THEN 240
100 IF G9='@' THEN 200
130 IF INDEX('#%$=, ',G9)#0 THEN 180
140                 REM   SPECIAL CHARACTER  Check special character
150 IF A(G8)#'%' THEN 180
153 IF INDEX('0123456789',G9)=0 THEN 180
160 G4=K(VAL(SUBSTR(G(G7),G3-1,2)))         Check keyword
162 IF H(G8)=G4 THEN 230
165 IF J(G7,5)#0 THEN 500 ELSE GOTO 40        - Failure to match production
180 IF A(G8)=G9 THEN 230 ELSE GOTO 165
195                 REM   POSSIBLE SUBROUTINE
200 G4=SUBSTR(G(G7),G3-1,1)
205 IF G4=' ' THEN 180 ELSE G6=G6+1
208 G5=G1
210 S(G6)=G7
215 G7=G(VAL(G4))
220 GOTO 50
230 G3=G3-2
235 G8=G8-1                                 Next picture character
237 GOTO 70
240 IF J(G7,2)#0 THEN GOSUB 301C              - Semantics?
250 IF J(G7,1)=0 THEN  41C ELSE G9=GS(J(G7,1)) - Replacement picture?
270 G4=LENGTH(G9)/2
280 J2=G4
```

```
290 IF RIGHT(G9,1)#' ' THEN 340
300 A(G8+G4)=A(G2)
310 H(G8+G4)=H(G2)
320 G4=G4-1
330 G9=SUBSTR(G9,1,2*G4)
340 IF G4=0 THEN 400
350 FOR G2=G8+1 TO G8+G4
360 A(G2)=SUBSTR(G9,2,1)
370 H(G2)=SUBSTR(G9,1,1)
380 G9=SUBSTR(G9,3)
390 NEXT G2
400 G2=G8+J2
410 IF J(G7,4)=1 THEN GOSUB 1010
415                   REM   SUCCESS
440 G4=J(G7,3)
445 J2='+'
450 GOTO 505
500 G4=J(G7,5)
502 J2='-'
505 IF G4=1000 THEN 550 ELSE IF G4=2000 THEN 560
510 IF G4<0 THEN 540 ELSE G7=G4
520 GOTO 50
540 PRINT '*** ERROR # ';G4, SUBSTR(GO,1,G1)
542 X
550 WRITE ON 'COPYFILE';BUT
552 GOTO 30
560 G7=S(G6)
565 G6=G6-1
570 IF J2='+' THEN 230
575 G1=G5
580 GOTO 165
1010 G2=G2+1
1030 G9=SUBSTR(GO,G1,1)
1040 IF G9#' ' THEN 1050 ELSE G1=G1+1
1045 GOTO 1030
1050 G4=G1
1055 IF G9= '  THEN 1330 ELSE IF G9=' ' THEN 1330
1060 IF INDEX(',;:',G9)#0 THEN 1190
1063 IF INDEX('+-*/ <>=#',G9)#0 THEN 1280
1065 IF INDEX('0123456789',G9)#0 THEN 1210
1070 IF INDEX('ABCDEFGHIJKLMNOPQRSTUVWXYZ',G9)#0 THEN 1110
1080 A(G2)=G9
1085 G1=G1+1
1090 GOTO 1170
1100                  REM   NAME
1110 G1=G1+1
1120 J1=SUBSTR(GO,G1,1)
1130 J2=INDEX('ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',J1)
1140 IF J2#0 THEN 1110
1165 A(G2)='%'
1170 H(G2)=SUBSTR(GO,G4,G1-G4)
1180 RETURN
1185                  REM   SEPARATOR
1190 A(G2)=','
1195 H(G2)=G9
```

Replace any match

Replace rest

- Scan -

- Failure branch -

- DONE! -

- Return from subroutine -

SCANNER SUBROUTINE

- Scan off blanks -

```
1197 G1=G1+1
1200 RETURN
1205                    REM   NUMBER
1210 G1=G1+1
1220 J1=SUBSTR(G0,G1,1)
1230 IF INDEX('0123456789',J1)#0 THEN 1210
1260 A(G2)='#'
1270 GOTO 1170
1275                    REM   OPERATOR
1280 G1=G1+1
1290 IF INDEX('<>=',SUBSTR(G0,G1,1))=0 THEN 1310
1300 G1=G1+1
1310 A(G2)='='
1320 GOTO 1170
1325                    REM   STRING
1330 J2=G9
1340 G1=G1+1
1350 IF SUBSTR(G0,G1,1)#J2 THEN 1340 ELSE A(G2)='$'
1360 G1=G1+1
1370 GOTO 1170
3000                    REM   SEMANTICS
3010 ON J(G7,2) GOTO 3100,3130,3145,3175,3195,3245,3142
3020 PRINT 'SEMANTICS ERROR'; J(G7,2),SUBSTR(G0,1,G1)
3030 GOTO 550
3095                    REM   SEMANTICS 1
3100 F=H(G2)
3102 IF VAL(F)<10 THEN 3020
3105 IF VAL(F) >=10000 THEN 3020
3115 OUT = ''
3117 F=1
3118 RETURN
3120 F=G1
3125 RETURN
3130          REM  SEMANTICS 2
3135 GOSUB 3400
3140 OUT=OUT+'0=0'
3141 GOTO 3120
3142 F=G1=LEN(H(G2))=1                    Semantics 7
3143 RETURN
3145                    REM   SEMANTICS 3
3147 GOSUB 3400
3150 OUT=OUT+'REM'
3155 GOTO 3120
3175                    REM   SEMANTICS 4
3177 GOSUB 3400
3180 OUT=OUT+'READ'
3185 GOTO 3120
3195                    REM   SEMANTICS 5
3197 GOSUB 3400
3200 OUT=OUT+'GOTO 10000'
3205 GOTO 3120
3245                    REM   SEMANTICS 6
3250 OUT=OUT+SUBSTR(G0,F,LEN(G0)-F)
3255 GOTO 3120
3400 J2=LENGTH(H(G2))                     COPY SUBROUTINE
```

```
3405 BUT=BUT+SUBSTR(GO,F,G1-F-J2)
3410 RETURN
5000 LINE= LBAD ' +FILE+'*'+'(ICTEST)'+ '
5010 WRITE BN 'CBPYFILE':LINE
5910 DATA 68,25,3,3          # productions, # keywords, # repl. pictures, # subroutines
6000 DATA '#','01','02','03','04','05','06','07','08','09' - production pictures
6010 DATA '01  ','01','01','01','01','01',' ','#',' ','1@'
6020 DATA '2@','3@','2@','23','#','%','#','$','(','=','( )'
6030 DATA '(  ',' ','2@',',',',','%',',',',','011C','01 @','01  ','11'
6040 DATA '12','13','14','15','16','17','18','19','3@','=','%','=','2@','20'
6050 DATA '2@','21','22','2@','13','13','24','25','=','(','1@',' ! )','1@'
6100 DATA 'IF','PRINT','DATA','RESTORE','INPUT','INPUTS','STOP','FALSE' - Keywords
6110 DATA 'END','ELSE','REM','FOR','GOTO','LET','NEXT','GOSUB','RETURN'
6120 DATA 'ON','SKIP','TO','STEP','BY','THEN','AND','OR'
6160 DATA '',' ',' |',34,26,36 - replacement pictures & subroutine entries
6200 DATA 1,1,2,1,-1,0,0,23,1,0,1,2,12,0,0,1,3,13,0,0,1,3,14,0,0
6220 DATA 1,4,15,0,0,1,4,15,0,0,1,5,16,0,0,1,5,16,0,0,1,5,16,0,0
6230 DATA 0,0,41,0,17,0,0,68,1,1000,0,0,20,1,17,0,0,21,1,17,0,0,22,1,17
6240 DATA 0,0,38,1,17,0,6,1000,0,0,-2,1,0,11,1,2,0,0,2000,0,0,0,0,38,0,38
6250 DATA 0,0,38,0,-5,0,0,38,0,-6,0,0,24,0,-7,1,0,25,1,61,1,0,38,1,2
6260 DATA 1,0,65,1,0,1,0,30,1,0,1,0,30,1,0,0,0,26,1,64,1,0,26,1,62
6270 DATA 1,0,30,1,0,0,0,26,0,33,0,0,2000,0,-10,0,0,35,0,2000,1,0,34,1,19
6280 DATA 1,0,37,1,-12,1,0,36,1,19,1,0,18,1,0,1,6,1000,0,0,2,0,2,0,-13
6290 DATA 1,0,38,1,0,1,0,52,1,0,1,0,21,1,0,1,0,50,1,0,1,0,22,1,0
6300 DATA 1,0,21,1,0,1,0,38,1,0,1,0,59,1,0,1,0,21,1,50,0,0,51,0,-14
6320 DATA 1,0,21,1,-15,1,0,53,1,-16,1,0,54,1,-17,0,0,55,0,-18,1,0,56,1,-19
6330 DATA 0,0,57,0,-20,1,0,21,1,0,1,0,21,1,38,0,0,60,0,-21,1,0,20,1,-22
6340 DATA 1,0,25,1,-8,1,0,26,1,0,1,0,26,1,31,1,0,26,1,-9,3,0,66,1,30
6350 DATA 0,0,67,0,-25,1,0,30,1,-26,0,7,38,0,-4
RUN
CLEAR
LBAD 'CBPYFILE'
```

productions (5 items per)

APPENDIX C
XBASIC PRODUCTIONS

Productions for processing XBASIC programs using TEST are given on the following pages. The rules for actually encoding these productions in XBASIC DATA statements are given in Appendix B, as are the encoded productions themselves a part of the TEST listing. What follows is a more readable representation of the productions.

As can be readily seen, READ FROM, WRITE ON, and multiple assignment statements separated by semicolons are not handled by these productions; but suitable productions could be added.

XBASIC PRODUCTIONS

prod #	picture	replacement picture	scan?	semantics	success branch	failure branch	
1	# >		*	1	2	-1	
2	IF		*	0	23	0	start new statement
3	PRINT >			2	12	0	
4	DATA >			3	13	0	
5	RESTORE >			3	14	0	
6	INPUT >			4	15	0	
7	INPUTS >			4	15	0	
8	STOP >			5	16	0	
9	PAUSE >			5	16	0	
10	END >			5	16	0	
11	IF ?			0	41	17	
12	IF		*	0	68	1000	
13	IF		*	0	20	17	
14	IF		*	0	21	17	
15	IF		*	0	22	17	
16	IF		*	0	38	17	
17	?			6	1000	-2	
18	# >		*	0	11	2	
19	?			0	2000	0	
20	1@			0	38	38	
21	2@			0	38	-5	
22	3@			0	38	-6	
23	2@			0	24	-7	
24	THEN >		*	0	25	61	
25	# >		*	0	38	2	
26	% >		*	0	65	0	expression subroutine (2@)
27	# >		*	0	30	0	
28	$ >		*	0	30	0	
29	(		*	0	26	64	
30	= >		*	0	26	62	
31	( ) >		*	0	30	0	
32	(?			0	26	33	
33	?			0	2000	-10	
34	2@			0	35	2000	expression list subroutine (1@)
35	, >		*	0	34	19	
36	% >		*	0	37	-12	name list subroutine (3@)
37	, >		*	0	36	19	
38	IF ELSE >		*	0	18	0	
39	IF @ >			6	1000	0	
40	IF ? >			0	2	-13	
41	REM >		*	0	38	0	start embedded statement
42	FAR >		*	0	52	0	
43	GOTO >		*	0	21	0	
44	LET >		*	0	50	0	
45	NEXT >		*	0	22	0	
46	GOSUB >		*	0	21	0	
47	RETURN >		*	0	38	0	
48	ON >		*	0	59	0	
49	SKIP >		*	0	21	50	
50	3@			0	51	-14	
51	= >		*	0	21	-15	
52	% >		*	0	53	-16	for statement
53	= >		*	0	54	-17	
54	2@			0	55	-18	
55	TO >		*	0	56	-19	

56	20		0	57	-20
57	STFP >	*	0	21	0
58	BY >	*	0	21	38
59	20		0	60	-21
60	CATB >	*	0	20	-22
61	CATB >	*	0	25	-8
62	AND >	*	0	26	0
63	BR >	*	0	26	31
64	= >	*	0	26	-9
65	(> [	*	0	66	30
66	10		0	67	-25
67	[ ) >	*	0	30	-26
68	10		7	38	-4