ED 258 556 IR 011 711

AUTHOR Putnam, Ralph; And Others
TITLE A Summary of Misconceptions of High School BASIC Programmers. Technology Panel Study of Stanford and the Schools. Occasional Report #010.
PUB DATE Aug 84
NOTE 24p.; For related document, see IR 011 707.
PUB TYPE Reports - Research/Technical (143)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Computers; *Error Patterns; High Schools; High School Students; Interviews; *Programing; Programing Languages; Psychological Studies; Screening Tests; *Testing; *Test Interpretation; Test Use
IDENTIFIERS *BASIC Programing Language; Misconceptions

ABSTRACT

Misconceptions high school students have about constructs in the BASIC programming language were examined in this study. A total of 96 high school students received a screening test after a semester course in BASIC programming, and 56 of these students were subsequently interviewed by means of questions and short programs prepared in advance. The purpose of the screening test was to detect possible problems with constructs such as reading data, branching, and looping. Tape recordings, written notes, and responses generated during the interviews were used to determine patterns of errors and misconceptions. Notations of frequently, fairly frequently, and occasional were given to the students' error rate. The most common errors were: (1) reversal of assignment statement; (2) misinterpretation of quotation marks; (3) difficulties with READ statements; (4) loop construction; and (5) IF statement utilization. Examples of the most common errors are included to illustrate each misconception together with student remarks as to why the error occurred. A summary assessment of students interviewed concludes the report. (JB)

OCCASIONAL REPORT # 010

Technology Panel

Study of Stanford and the Schools

A Summary of Misconceptions of High School BASIC Programmers

Ralph Putnam
Derek Sleeman
Juliet Baxter
Laiani Kuspa

August 1984

These occasional reports present preliminary findings of research underway or discuss issues of concern to the panel. They are intended to stimulate comment and to maintain communication with interested parties both within and outside the Study of Stanford and the Schools. They are draft documents and are not to be quoted or cited. They do not necessarily represent the views of Stanford, the Study as whole, or even the panel as a whole. Members of the panel and addresses where they may be reached are included at the rear of the report. Comments of readers will be greatly appreciated.

A Summary of Misconceptions of High School BASIC Programmers

Ralph T. Putnam
D. Sleeman
Juliet A. Baxter
Laiani K. Kuspa

School of Education
Stanford University
Stanford, CA 94305

Abstract

This study examined the misconceptions students in high-school programming classes had about constructs in the BASIC programming language. A screening test was administered to 96 students, 56 of whom were interviewed. Students were asked to trace simple programs and predict their output. Errors in virtually all BASIC constructs we examined were observed.

Introduction

This paper reports the misconceptions students in high-school programming classes had about constructs in the BASIC programming language. The background and justification for such a study have been laid out in an earlier paper (Sleeman et al., in press). In this paper we describe the procedures we used and present a summary of the misconceptions and difficulties we encountered with students who had studied BASIC.

Method

Subjects

Students from five high-school classes participated in the study. The first class (9 students) was designed to teach mathematical concepts by using BASIC programming; little emphasis was placed on developing programming style and competence. Three classes (64 students) were second semester courses in BASIC programming. Students in these classes were interviewed near the beginning of the school term; all had completed an introductory BASIC programming course. The fourth class (23 students) was a first course in BASIC programming. Students in this class were interviewed near the end of the school term. Of the total of 96 students who took the screening test (described below), 56 were interviewed.

Screening Test

A test of BASIC programming concepts was developed for the study. The test was devised after examining programs written by students in one class and probing a few students about programs they had written. The purpose of the test is to detect possible problems in basic constructs such as reading data, branching, and looping. Items 1 to 4 require writing the output produced by very short (four to ten line) programs, each designed to highlight a single concept. The remaining items consist of slightly more complex programs. The task for two of these items (6 and 9) is to debug a program for which a written description of the intent of the program is provided. The student must determine the output to slight variants of programs in two pairs of items (5,7 and 8,10).

The order of items 5 through 10 was varied in two different forms of the test to insure coverage of the topics if students were not able to complete the test in the time available. Students were asked to rate the difficulty of each item to help determine the appropriateness of the items.

Experimental Procedure

The screening test was given to volunteer students in the first class and to all students in the other four classes. Each student's performance was evaluated by one of the researchers who decided whether the student should be interviewed, was a marginal candidate for an interview, or did not need an interview (because the student had minor or no difficulties, or manifested a well understood set of misunderstandings). The interviews were clinical in nature, with interviewers using questions and short programs prepared in advance, but also following up with various probes and programs composed on the spot. The goal was to clarify as far as possible the nature and extent of the students' misconceptions about programming concepts. Students were asked to say what output would be produced by various programs, to trace programs and explain how they work, and to debug short programs. In several cases, students were asked to trace identical programs with different sets of input data. The questions generally continued until the researcher was able to decide: i) the nature of the student's error, ii) that the student had a variety of possible ways of interpreting a construct or iii) that the student had little knowledge of a particular concept. For further details of this overall methodology, see Sleeman (in preparation). Many of the programs and program fragments used in the interviews are presented in the text. A complete set of the materials used will be furnished upon request by the authors.

Tape recordings, written notes and responses generated during the interviews were perused for patterns of errors and misconceptions. As the study was exploratory and qualitative in nature, no quantitative analysis techniques were used. Findings are discussed in the following section.

Summary of Misconceptions Encountered

A Comment on the Frequency of Errors

As noted above the screening test was given to 96 students of which 56 were subsequently interviewed. We shall refer to a misconception as being frequent with this population if it occurred with 25% or more of the interview population (i.e. 14 or more students), fairly frequent if it occurred with 6-13 students, and occasional if it occurred less frequently (i.e. with 1-5 students). These figures do not reflect the frequency or the consistency with which each misconception was used by individual students; specific comments about these aspects will be interspersed throughout this section.

Assignment Statements

Reverse assignments. The most common assignment error was the "reversal" of an assignment statement. For example, the statement LET A = B was thought to assign the value of A to B rather than the value of B to A. Students making this error generally interpreted statements such as A = B + C correctly. This was an occasional error.

Counter. One student declared that the statement LET C = C + 1 was impossible. He had previously assigned the value of 0 to C and interpreted the statement as "Let 0 equal 0 + 1." He said thir did not make sense and seemed to think that the statement was an error. Even when the interviewer had the student work through a program containing the statement LET W = A * 0, which the student interpreted correctly, the student could not make sense of LET C = C + 1.

PRINT Statements

Quotation marks. Three students misinterpreted quotation marks in PRINT statements such as PRINT "Q:"; Q. One student simply ignored the text within the quotation marks, printing the value of Q one time. Another student thought the quotation marks caused the value of the variable to be printed, saying that the statement would print 4:4. A third student thought that the quotation marks would cause the first value assigned to the enclosed variable during program execution to be printed, resulting in the output 0:4 for the statement PRINT "Q":Q (LET Q=0 was the first statement in the program). This last error was the only one of the three that suggested a deeper misconception that a variable could "remember" its original value.

Repeated print. One student, who had major programming difficulties, thought that the statement PRINT X would cause the value of X to be printed several times -- enough times to fill up about half a line on the screen. The student gave similar outputs for seven different programs. Although he was not entirely consistent, he generally said the value would be printed only once if there were more data items to be read by the program, several times if there were not more values to be read.

Multiple-value print. Some students thought that when a variable was printed, all the values that had been contained in that variable were printed. This occasional misconception is related to <u>multiple-valued variables</u> and will be discussed below.

<u>READ Statements</u>

More of the students we interviewed had difficulties with READ statements than with any other aspect of the BASIC language. These difficulties were evidenced both by the large numbers of students for which they occurred and the large number of different errors made. All of the students interviewed had seen and used READ statements, although many of them made heavier use of other constructs for inputting data (i.e., INPUT statements). Nevertheless, reading data seemed to be a difficult concept for these students.

<u>Feature-dependent reads</u>. Several of the errors with READ statements involved the belief that the program can select values from the DATA statement on the basis of features of those values (the values are actually read sequentially).

1. We call the most common of these feature-dependent reads <u>semantic read</u>. Students with this frequent misconception believed that a READ statement used with a meaningful variable name caused the program to select a value based on the name's meaning from the list of values in the DATA statement instead of reading the next value in sequence. For example, given the following program fragment:

```
40 READ SMALLEST
50 READ FIRST
60 READ SECOND
70 READ THIRD
80 READ FIRST
200 DATA 99,2,-3,-100,6,29
```

these students said that -100 would be read into SMALLEST, 99 into FIRST, 2 into SECOND, -3 into THIRD, and then 99 into FIRST. Semantic constraints created by the meaningful variable names determined which values were read from the line of data. Most of the students with this misconception were consistent, incorrectly assigning semantically constrained values to all meaningful variable names encountered.

In a variant of the semantic read misconception, three students at first appeared to be using variable names as constraints but probing revealed that they believed the meaningful variable names functioned as subroutine calls or branches to other parts of the programs. One of these students realized later in the interview that he was indeed dealing with variables and subsequently interpreted the READ statements correctly.

2. Some of the students with the semantic read misconception also tried to impose meaning on single-letter variable names in READ statements. In an occasional misconception, some students thought the position of the letter in alphabetical order determined the position of the value in the data line to be read. In the program below:

```
40 READ A
50 READ B
60 READ N
200 DATA 9,38,-100,5,12
```

these students said 9 would be assigned to A (because 9 is the first value in the DATA statement), 38 (the second number) would be assigned to B, and 12 would be assigned to N (because N is near the end of the alphabet and 12 is the last number in the list of data). In another program in which N was the only variable used, one student again said N would be assigned the last value in the DATA statement. She said this value was assigned to N in _each_ of the three READ N statements.

3. Some students thought that values assigned to single-letter variable names were constrained by other statements in the program. For example, when confronted with the following program:

```
40 READ A
50 READ B
60 READ N
200 DATA 9,38,-100,5,12
```

one student stated that he could not say what values would be read into variables A, B, and N because he did not know what values the variables were supposed to have. The interviewer then added lines to make the program read as follows:

```
10 LET A = 5
20 LET B = 38
30 LET N = 9
40 READ A
50 READ B
60 READ N
200 DATA 9,38,-100,5,12
```

The student then said that at line 40 the 5 from the DATA statement would be read into A, at line 50 the 38 would be read into B, and at line 60 the 9 would be read into N.

Another student believed that IF statements exerted similar control over READ statements elsewhere in the program. He said that at line 10 of the following program, N would "pick" 0 from the data line because of the condition in line 30:

```
10 READ N
20 PRINT N
30 IF N <= 0 THEN GOTO 20
40 DATA 34,3,16,10,0
50 END
```

Multiple-value read. In a frequently occurring misconception, students thought that a READ statement could cause more than one value to be assigned to a variable. This bug often occurred in conjunction with semantic read as in this program:

```
40 READ EVEN
50 READ ODD
60 READ POSITIVE
70 READ NEGATIVE
200 DATA 9,38,-100,5,12
```

Students said that the values 38, -100 and 12 would be read into EVEN, 9 and 5 into ODD, and so forth.

Other students thought that all the values in the DATA statement were read into a variable. For example, in the following program:

```
10 READ X
20 READ Y
30 IF X = 0 THEN GOTO 80
40 IF X = 1 THEN GOTO 60
50 PRINT X
60 PRINT Y
70 GOTO 10
80 END
90 DATA 2,3,1,2,-5,-9,0
```

students said that all the values would be read into X and into Y. One student subsequently had difficulty interpreting the IF statement in line 30. As this statement required evaluating only one value and X contained several, the student was unable to predict what the program would do.

In a final variant of multiple-value read, some students thought that the number of characters in the variable name determined the number of values read, with one value being read into each character.

Read control. Several misconceptions of the READ construct involved the way in which the assignment of values to variables was controlled or ordered. The first of these misconceptions was fairly frequent; the rest were occasional.

1. Multiple READ statements containing the same variable repeatedly accessed the same value in the line of data. Given the following program:

```
40 READ N
50 READ N
60 READ N
200 DATA 3,6,9,12
```

students with this misconception said that 3 would be assigned to N by each of the READ statements.

2. READ statements appearing after all the values in the DATA statement have been read continue to access the last value in the DATA statement.

3. When the list of data values is exhausted, further READs cause the value 0 to be assigned to variables.

4. When the list of data values is exhausted, further READ statements go back to the beginning of the list of values. In some students, this misconception occurred with multiple value reads as in the following program:

```
40 READ N
50 READ N
60 READ N
200 DATA 3, -6, 9, 12
```

At least one student said that all the values (3,-6,9, and 12) would be read into N at line 40, again at line 50, and again at line 60.

5. Each variable in a program reads the line of data independently. In the following program:

```
10 READ X
20 READ Y
30 IF X = 0 THEN GOTO 80
40 IF X = 1 THEN GOTO 60
50 PRINT X
60 PRINT Y
70 GOTO 10
80 END
90 DATA 2,3,1,2,-5,-9,0
```

students with this misconception said that X would be assigned 2 and Y would be assigned 2. On the second time through, X would be assigned 3 and Y would be assigned 3, and so on forth.

6. A separate DATA statement is required for each READ statement. The program shown above would not work unless a second DATA statement were added.

7. A READ statement causes the value to be read to be added to the original value in the variable. Because the student with this misconception also thought that READ statements must be constrained by other statements in the program, a program was modified as follows:

```
10 LET N=9
40 READ N
50 READ N
60 READ N
200 DATA 3,-6,9,12
```

The student said that at line 40 N would read the 9 (because of the constraint in line 10). At line 50, the 3 would be added to 9 to make the value of N 12, and at line 60, the -6 would be added to 9 to make the value of N 3.

8. The user selects the value to be read by the READ statement. The student with this misconception was not sure how the user would select a value. He was apparently confusing READ statements with INPUT statements.

Variables

Multiple-valued variables. The most significant misconception involving variables was that a variable can contain more than one value. These multiple values occurred in a variety of ways. As discussed earlier, some students believed that several values could be assigned to a single variable by a READ statement (multiple-value read). Other students knew that values were read one at a time but thought that the values were collected in the variable as they were read or assigned (like a stack). A PRINT statement would then cause all of the values in the variable to be printed.

The following additional difficulties involving variables were seen occasionally:

1. Two variables are confused. For example the following lines of a program:

```
...
20 READ P
...
50 LET Q = Q + 1
...
```

students interpreted line 50 as if it were LET Q = P + 1.

2. The values of variables are not updated; the initial value is used repeatedly. This error occurred in the following program:

```
10 READ SMALLEST
20 READ N
30 IF N = 0 THEN GOTO 60
40 IF SMALLEST > N THEN LET SMALLEST = N
50 GOTO 20
60 PRINT SMALLEST
70 DATA 2,1,3,4,0
```

At line 40 the students continued to compare the most recent number read into N to the value 2 in SMALLEST, even after correctly changing the value of SMALLEST to 1. This error seemed to be an instance of a general difficulty in keeping up with the values of variables when tracing programs.

Loop construction

Several errors involved loop constructions. The errors that are not limited to a particular kind of loop will be discussed first, followed by those specific to FOR/NEXT loops. Unless otherwise noted, these errors appeared occasionally.

1. A PRINT statement following a loop is repeated as though it were inside the loop. For example, in this program:

```
10 LET Q = 0
20 READ P
30 IF P = 0 THEN GOTO 70
40 IF P < 0 THEN GOTO 60
50 LET Q = Q + 1
60 GOTO 20
70 PRINT "Q:"; Q
80 DATA 1,-1,2,5,-4,-6,10,-3,0
90 END
```

students said the value of Q would be printed each time the loop was executed. This error occurred fairly frequently.

2. <u>Data-driven looping</u>. The loop iterations continue as long as there are data to be read. For example, for the folowing program:

```
10 FOR I = 1 TO 5
20 READ X
30 PRINT X
40 NEXT I
50 DATA 5,8,6,3,10,11,1,25,2
```

the following predicted output was observed:

```
5   8   6   3   10   11   1   25   2
```

3. A unique error involving a loop appeared with the following program:

```
10 READ A
20 READ B
30 IF B = 0 THEN GOTO 60
40 IF A > B THEN LET A = B
50 GOTO 20
60 PRINT A
70 DATA 55,6,3,-2,0
80 END
```

The student returned to line 10 instead of line 20 for each iteration of the loop, thus reading a new pair of values into A and B instead of a single value into B. He also accessed the values in the data statement incorrectly so that the pairs of values he read were 55,6; 6,3; 3,-2; and -2,0.

4. Misconceptions specific to FOR/NEXT loops.

a. The FOR statement acts as a constraint on READ statements within the loop rather than determining the number of times the loop body is repeated. This fairly frequent bug is illustrated in the following program:

```
10 FOR I = 1 TO 5
20 READ X
30 PRINT X
40 NEXT I
50 DATA 5,8,6,3,10,11,1,25,2
```

Some students said only the values between 1 and 5 would be read, resulting in the following output:

```
5   3   1   2.
```

Other students said the values would be selected in order of the constraints, resulting in the following output:

```
1   2   3   5.
```

Finally, in a program which was the same as the one above except for the DATA statement:

```
50   DATA 5,4,3,2,1,0,1,2
```

these two predicted outputs were observed:

```
1.   5   4   3   2   1

2.   5   4   3   2   1   1   2
```

b. The FOR statement specifies the number of times a variable's value should be printed when it should determine how many values are read and printed. The following program from the screening test was involved with this occasional error:

```
10 FOR I = 1 TO 5
20 READ X
30 PRINT X
40 NEXT I
50 DATA 5,8,6,3,10,11,1
```

The student predicted the following output:

```
5    5    5    5    5
8    8    8    8    8
6    6    6    6    6
3    3    3    3    3
10    10    10    10    10
11    11    11    11    11
1    1    1    1    1
```

c. Some students did not realize that the counter variable in a FOR statement is a variable that is incremented with each iteration of the loop. These students thought that it was acceptable to change the value of the counter variable within the loop body.

IF statements

Four occasional misconceptions involving IF statements wre observed.

1. Execution of the program terminates when the condition of an IF statement is not true.

2. Control is passed to the beginning of the program when the condition of an IF statement is not true.

3. A program must state all possible conditions when using IF statements. One student "corrected" the following lines of a program:

```
...
40 IF X >= 70 THEN GOTO 60
50 LET I + I + 1
```

She changed line 50 to read:

```
50 IF X < 70 THEN LET C = C + 1.
```

She argued that the IF statement in line 50 was necessary so the computer would know that X could be less than 0.

4. When an IF statement sends you to a PRINT statement, print both the variable to be printed <u>and</u> the value in the conditional expression. The following lines of a program were involved in one instance of this misconception:

```
30 IF N = 0 THEN GOTO 60
...
60 PRINT SMALLEST
```

The student said the output would be:

```
1 0
```

When the conditional in line 30 was changed to N= -99, the student said the output would be:

```
1 -99
```

The student predicted similar output for other programs.

Other Flow of Control Difficulties

In addition to the difficulties involving loops and IF statements, we observed at least two occasional difficulties with the flow of control in programs.

1. All PRINT statements are executed (even if they should be skipped because of a branching statement).

2. All statements in a program must be executed at least once, even statements that might be skipped because of branches in the program. When one student was asked to trace the following program:

```
10 LET X = 1
20 LET Y = 2
30 IF X = 1 THEN GOTO 50
40 PRINT X
50 PRINT Y
70 END
```

the student gave a correct interpretation through line 30 and said correctly that 2 would be printed at line 50. At that point, however, the actual end of execution, she said that because line 40 was missed, the computer would go back to line 40 and print X, then continue to line 50 and print Y a second time. She said that execution now ended "because all the statements had been visited."

Tracing and Debugging

In addition to the particular misconceptions or bugs we have described, many students had more general difficulties tracing and debugging programs. Most students were asked to trace programs during the interviews and some were given a program to debug. We saw the following sorts of difficulties:

1. Students inferred the function of a program from a few statements. They would trace or predict output ignoring or misinterpreting statements that did not fit with the way they thought the program should work.

2. Students concentrated on small segments of the program when debugging, making assumptions about what other parts of the program did.

3. Students had difficulty keeping track of the values of variables (reflected in an error described earlier in the section on variables).

Summary Assessments

The interviewers classified each student as having essentially no difficulties, minor difficulties, or major difficulties. The ratings are tabulated in Figure 1. Several students had no difficulties in the interview, although they made errors on the screening test. In most cases these students made careless errors or rushed through the test. Of the students with difficulties, equal numbers are classified as having major and minor difficulties.

Figure 1

Summary Assessments of Students Interviewed

	Male	Female	Total
No difficulties	11 (32%)	1 (6%)	12 (24%)
Minor difficulties	11 (32%)	8 (50%)	19 (38%)
Major difficulties	12 (35%)	7 (44%)	19 (38%)
Total	34	16	50

Note. Summary assessments were missing for 6 students.

To present a better picture of the summary classifications, we will describe all the difficulties noted for two students. One was classified as having minor difficulties, the other had major difficulties.

Student with Minor Difficulties

This students made two kinds of errors:

1. The student gave semantic read interpretations in four programs. Although the student was consistent in selecting values from the DATA statements based on the meaning of the variable names, she was not entirely confident with her responses as she had not used meaningful words for variable names before.

2. In one program the student said multiple READ statements containing the same variable repeatedly accessed the same value of the data statement. After she was given a variant of the program containing a loop, she corrected her interpretation of the program.

Student with Major Difficulties

This student made several errors revealing considerable misunderstanding of BASIC constructs.

1. The student gave semantic read interpretation to READ statements.

2. The student refused to interpret READ statements containing single-letter variable names until the variables were given some meaning or value. In the following modified program:

```
10 LET N = 9
40 READ N
50 READ N
60 READ N
200 DATA 3,-6,9,12
```

the student said that each of the READ statements would read the 9 from the line of data. This error occurred in two programs.

3. The student said that a READ statement would read an entire line of data when it appeared within a FOR/NEXT loop. For this program:

```
10 FOR I = 1 TO 5
20 READ X
30 READ X
40 NEXT I
50 DATA 5,8,6,3,10,11,1,25,2
```

he gave the following output:

```
5,8,6,3,10,11,1,25,2
5,8,6,3,10,11,1,25,2
5,8,6,3,10,11,1,25,2
5,8,6,3,10,11,1,25,2
5,8,6,3,10,11,1,25,2
```

The student was quite confident in this interpretation.

4. In interpreting one program the student thought the program would continue as long as there were more data values to be read. When

probed about this, he thought the looping was controlled by the READ statement.

Summary

We have reported numerous misconceptions held by high-school students in beginning BASIC programming courses. Errors were found with virtually every construct included in all tests and interviews. As in our work with Pascal (Sleeman et al., in press), students with a semester or more of experience with BASIC had very fuzzy knowledge about how various constructs operate; their knowledge of the conceptual machine underlying BASIC was weak.

ACKNOWLEDGEMENTS

We wish to thank the teacher and students who participated in this study. Haym Hirsch and Alan Char helped with the interviewing. Marcia Linn helped by providing access to an earlier test of BASIC programming. We also thank the Study of Stanford and the Schools, directed by Dean M. Atkin and President Kennedy, and D. Walker and M. Lepper (Chairs of the Panel on Education and Technology) for providing funding for the study. Finally, Mrs. Dorothy Brink provided invaluable secretarial support.

REFERENCES

Sleeman, D., Putnam, R. T., Baxter, J. A., & Kuspa, L. K. (in press). Pascal and high-school students: A study of misconceptions.

Sleeman, D. Protocol Analysis & Interviewing: some methodological issues.

Footnotes

1. Also Computer Science Department.