4-1-1985

The effects of various error messages and error types on program debugging time.

David L. Dawson

THE EFFECTS OF VARIOUS ERROR MESSAGES AND ERROR TYPES ON PROGRAM DEBUGGING TIME

A Thesis

Presented to the

Department of Psychology

and the

Faculty of the Graduate College

University of Nebraska at Omaha

In Partial Fulfillment

of the Requirements for the Degree

Master of Arts

University of Nebraska at Omaha

by

David L. Dawson

April, 1985

Accepted for the faculty of the Graduate College, University of Nebraska at Omaha, in partial fulfillment of the requirements for the degree Master of Arts.

Graduate Committee

Name	Department
Evan Brown	Psych
[illegible]	Psy
John P. Maloney	Math/CS

[illegible] Psych

Chairman

April 19, 1985

Date

Acknowledgements

First, and most importantly, I wish to express my gratitude to my wife, Leigh Anne, for her unending understanding, support, and hard work. Without her uncomplaining financial and emotional support I certainly could not have finished this project.

I must express a deep admiration and appreciation to Dr. Evan Brown. Many times and in many situations during my studies at the University of Nebraska at Omaha Dr. Brown has supported and assisted me. I am grateful for the personal interest that resulted in my beginning graduate study, for the time taken to argue my case so that I could get an assistantship, and for so many small kindnesses that made me feel like part of the psychology department.

My parents certainly must share a part of this as well, for feeding me when I struggled to finish my graduate studies; but beyond this for teaching me to never say "can't" and to believe in myself.

I must also express appreciation to Dr. Deffenbacher who gave me many insights while I prepared this paper and was able to organize someone as disorganized as I.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

Using the Shneiderman model, programs were bugged with three classes of errors and then flagged with five classes of error messages both to test the the Shneiderman theory and to attempt to develop more useful messages for programmers.

Forty subjects, all of whom passed a general selection quiz on the programming language BASIC, participated in the study. Each subject saw three versions of a BASIC statistics program listing, the versions represented the three different types of errors and all shared a common sort of error message.

The error types strongly affected the time subjects required to find the bug in the printed listing and correct it. As predicted by the theory, the syntax error was found most quickly. However, the speed at which the flow error and misuse error were detected was reversed from the order predicted by the theory. Using the scores on the BASIC quiz as a covariate, the error messages were found to affect the time to debug a listing with knowing which line caused the error reducing the time most but having a greater effect on the syntax error over the flow error.

Though computer user errors have been studied in both controlled experiments and in less obtrusive designs, the error message, which has the potential to improve the system directly, has received little attention. Using as a frame of reference the Shneiderman model (Shneiderman, 1980), certain error messages were selected from suggestions made in the literature and these, along with the errors that created the messages, become the focus of this investigation.

As in other areas in human factors research, many of the published papers regarding computer-human interaction tend to be problem specific. For example, there are papers such as one by Geiselman and Samet (1982) regarding a specific sort of army intelligence message. Another paper discusses patients' attitudes regarding a system that obtains medical histories (Lucas, 1977). Historically, certain specific issues have dominated the human-computer interaction literature. For example, the controversy over the superiority of batch versus on line systems (e.g. Pollack, 1976) or the dispute about the best and safest video display (e.g. Baron, & Levenson, 1977; Helper, 1976; Huchingson, Williams, Reid, & Dudek, 1981). There are also studies limited to system specific problems (e.g. Bevan, 1981; Hansen, 1976; Jacob, Egeth & Bevan, 1976; Janelle & Polis, 1980; Rouse, 1977).

In addition to the limitations resulting from extreme specificity, quite a few studies lack empirical support. Many present a system that the author has just constructed and mainly serve as a place to show off this new piece of software (for examples see Boies, 1974; Helander, 1981; Singer, Ledgard, & Hueras, 1981; Treu, 1982; Zloof, 1982).

To pull together these diverse studies a number of theoretical frameworks have been suggested -- many converted from popular psychological theories. Peace and Easterby (1973) suggest using notions about constructs developed in psycholinguistics to aid in our understanding of human/computer dialogs. Using another converted theory, Oberquelle, Kupka, and Maass (1983) suggest a human communication model to help understand human communication with computers.

The prevailing theory is the Shneiderman model (also known as the Shneiderman/Meyer model), an obvious variant on a common view of cognitive psychological theorists concerning human functioning. The model is shown in Figure 1 along with my view on how this might be contrasted with the common cognitive model. The most important two concepts in this view are semantic knowledge and syntactic knowledge (Shneiderman, 1980). Semantic knowledge is defined by Shneiderman as low level concepts shared by most programming

	memory structures			long term memory (LTM)		
conventional information processing theory	sensory memory	short term memory STM		episodic memory (history)	semantic memory (facts)	procedural memory (skills)
Shneiderman Model	"input" from perception	short term mem (STM)	working mem	episodic memory (not mentioned --not relevant to the model?)	semantic memory (only part of entire semantic memory) (general concepts)	syntatic memory (only part of entire procedural) (syntax rules)

Figure 1. Structures of the Shneiderman model and the general cognitive model

languages. An example would be an understanding of what an assignment statement does. Syntactic knowledge is then the precise way that a specific language expects each statement to be phrased.

This view is not really that divergent from a common cognitive view (e.g. Bourne, Dominowski, & Loftus, 1979; Seamon, 1980), with Shneiderman's semantic knowledge being only a small part of what is normally labeled as semantic knowledge. But syntactic knowledge, I feel, translates to part of what is commonly called procedural knowledge by cognitive theorists as this is only the skill of producing the correct set of symbols (see Figure 1).

Many of the findings concerning errors in systems seem to confirm the existance of these two different structures in program generation. Several descriptive studies, that simply recorded and described errors that occurred on existing systems, found that some errors are corrected quickly while some others take quite a bit longer. Lang, Lang, and Auld (1981), looking at errors in a batch environment, found that 70% of the errors made in programs were found right away while 6% of the errors required more runs. In a study of errors made by users of an SPSS statistics package (Davis 1983), the researchers found two types of errors which they called "quick" errors and "slow" errors. Gade, Fields, Alison, Maisano, & Marshall (1981) found that 7% of the errors made by army data entry personnel seemed to be what they called "cognitive" errors that normal error corrector software could not help with. Gould and Drongowski (1974) bugged a series of programs and showed that errors that required the programmer to understand the deeper structure of the program took significantly longer to find than those that were simple errors of syntax. Authors of error correctors (Fischer & Mauney, 1980; Tai, 1978) have often noticed a different sort of error that cannot be repaired by simply comparing correct syntax to a line containing an error. Several other studies also suggest that there are typing or simple errors and something else (semantic errors) (Ehrenreich, 1981; Ripley & Druseikis, 1978).

It would seem that the Shneiderman theory would predict these various results with the "quick" errors being errors of syntax and the "slow" errors ones of semantics. In the Shneiderman book, cited earlier, the author further links his theory to these findings by hypothesizing that debugging a program involves rethinking the author's program generation process in reverse -- understanding the syntax first, then the program flow. This would necessitate more time be taken to find errors earlier in the thought process.

Arblaster (1982) faults the Shneiderman model for being too simplistic and not showing the greater amount of integration that he feels is present in any cognitive system. In answer to this charge, I believe that Shneiderman's model represents only the structures that he feels are present in the system, not the process involved. Later in his book Shneiderman discusses this point, giving the mental process of program generation which is shown in Figure 2. This process uses the structures shown in Figure 1. I have added what I feel is the material expression of each of these stages and the structures in long term memory that are most involved at each stage. This diagram also indicates the error types discussed in the Shneiderman book, and where I believe they occur in this thought process.

These three kinds of errors are, listed from the beginning of the thought process to the finished product: (1) errors that occur in the transformation from problem solution to semantics -- hereafter called Flow errors, (2) errors in transforming the semantics to syntax (Misuse errors) and (3) errors in typing the statements into the computer (Syntax errors). The last error type would commonly be called a typing error, the second an error in understanding the function of a statement, and the first an error in the strategy used to solve the problem, requiring

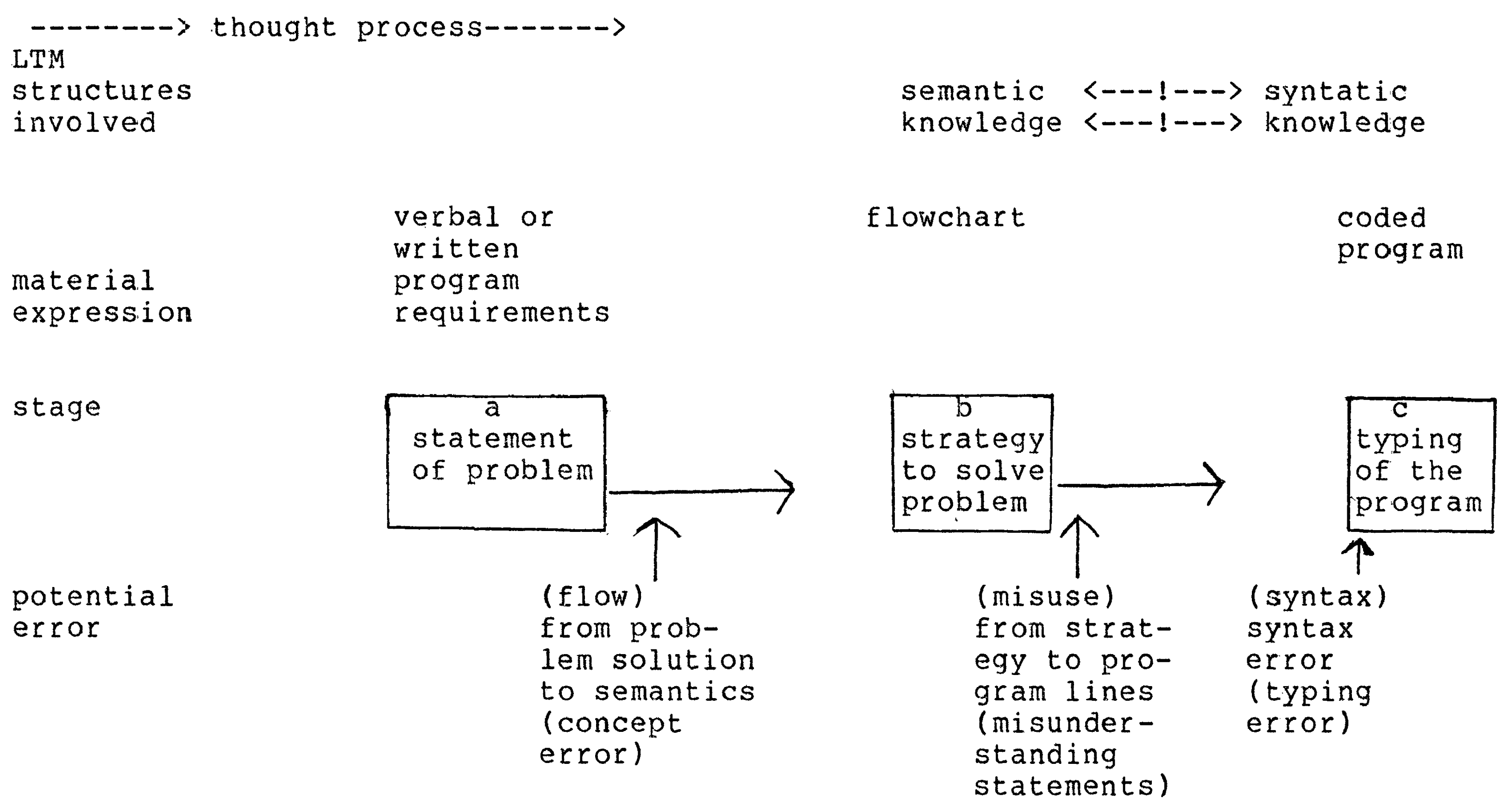
--------> thought process------->
LTM structures involved
semantic knowledge <---!---> syntatic knowledge
material expression
verbal or written program requirements
flowchart
coded program
stage
a statement of problem
b strategy to solve problem
c typing of the program
potential error
(flow) from problem solution to semantics (concept error)
(misuse) from strategy to program lines (misunderstanding statements)
(syntax) syntax error (typing error)

Figure 2. The thought process in program generation

perhaps a complete program rewrite.

Error classification schemes exist for many other tasks as well as that of computer programming (Rouse & Rouse, 1983; Huchingson, 1981; Reisner, 1977) and although the present scheme follows the Shneiderman model generally, there is really no data to justify the existence of more than the simple syntax error and some other "deeper" sort of error.

Findings in feedback studies were also consulted when considering the error classification scheme to be used to generate the specific errors to be studied here (Gleitman, 1981; Wexley & Yukl, 1977), as error messages are really a form of feedback. Unfortunately, the recommendations made by feedback theory are usually limited to suggestions that the feedback be as specific and immediate as possible. This seems to lack the explanatory power of the cognitive theories, especially with the complex task of program construction.

No systematic experimentation has been applied to error messages for programmers. The Gould and Drongowski (1974) article mentioned earlier has come the closest. In their exploratory study of program debugging, they provided varying conditions for programmers, including sample output and the line number of the problem line. This line number information did reduce the time to find the bug by half, unlike the assistance of sample output. Mostly though, the suggestions on error messages have been vague and limited to suggesting that an error message should be "understandable, non-threatening, and low-key" (e.g. Maguire, 1982).

Huchingson (1981) cites a Pew and Rollins study that bases several error message construction suggestions on "the author's background and limited interviews with the Department of Agriculture system analysts." These authors suggest that the location of the error, its nature, and a suitable recovery procedure be given. Other authors (McDonald & McNally, 1982; Nickerson, Elkind, & Carbonell, 1968; Sondheimer & Relles, 1982) all make suggestions relating to error messages or help functions, but in all of these, as in much of the literature, no data were collected. Prescriptions were made on the basis of some head scratching by the people in the data processing department.

In the proposed study three sorts of errors will be constructed based upon the Shneiderman model. These errors will then be flagged with several different sorts of error message and the time to find and correct each error will then be measured. The expectation is that the error message dealing with the stage of thought the error came from should be most effective in assisting the programmer in finding it and correcting it. BASIC will be used as it is the most often encountered language by newcomers and it has been shown to be easier to write in than FORTRAN (Jutila & Baram, 1971).

Many theorists seem to believe that only syntax errors produce error messages (e.g. Gould & Drongowski, 1974). Though the error messages produced by Flow or Misuse errors are certainly somewhat misleading (see the listings in the Appendices) it is possible to get indirect error messages from these errors. Many error messages may appear when running a syntactically correct program. Some examples are overflow, division by zero, bad subscript, functional call, and type mismatch errors. Most of these situations occur when data values become too large or take the wrong form. These are definitely not problems that a compiler would detect (which has been suggested as a definitional test for syntax errors) but the events still produce error messages.

In designing this study much thought was given to the points brought forward by Moher and Schneider (1982) in their paper dealing with experimental methodology of computer human factors research. I believe I have avoided the problems they discuss in subject selection. Mainly, there have been questions concerning the possibility that novice and experienced users are enough different to make generalization very difficult. This appears as a major concern when researchers use novice undergraduates to make suggestions about programming techniques. By using a pretest to screen for persons with a certain level of ability in BASIC programming and by defining the subject population as such I feel I have reduced this problem. I have avoided problems in language selection by using BASIC, a very common microcomputer language, instead of some experimental and unknown language.

Several predictions about the effectiveness of certain error messages can be made using the Shneiderman model, and this theoretical model guided the selection of the particular error messages included. The first prediction is that the message that shows where on a line the computer stopped parsing should help most with the simple syntax error. Generally, this should be the case because the message putting its "electronic finger" on the misspelled keyword should immediately pinpoint simple typing errors. Errors created by misunderstanding the statement used (Misuse errors) should be less clarified by just pointing to where the computer stopped because this kind of message does not instruct how the statement should be used. Similarly, errors in missing code and other general problem solving strategy errors (Flow), should not be helped as much by just knowing where the computer stopped on a particular line.

Secondly, reminders of command syntax in a problem line should help most where a given statement is not understood by the programmer (Misuse error) than when the error is problem solving strategy or simple typing error.

Finally, messages that give a listing of the last few lines run and that dump the value of variables when the program stopped should be of little help if a simple typing error was made (Syntax) or the function of a statement is misunderstood (Misuse) but should give valuable information about the overall program functioning and the effectiveness of the problem solving strategy used (Flow errors).

Method

Subjects and Design.

The experimental design was a simple 3 X 5 factorial, the first variable concerning three error types (syntax, misuse, and flow) and the second variable five types of error messages (see Appendcies C through Q). Error type was within subjects, while message a type is a between subjects.

Since there are 5 between-subject conditions in this study and inasmuch as a minimum of 8 subjects per cell were desired, 40 subjects were selected from three computer user groups in the Omaha Nebraska area (the TRS-80 Color Computer user group, the TI99/4a user group, and the VIC-20 user group) and the undergraduate population at the University of Nebraska at Omaha, including psychology students, and computer science undergraduates. Some professors in the Psychology Department and graduate students that were capable in BASIC were also called upon. Even with these divergent groups it was quite difficult to find a sufficient number of programmer-level subjects. Course extra credit was offered to all university subjects as inducement, and computer programs written by the author were offered to all computer group members. All of the subjects were randomly assigned to conditions. All participants had to reach the criterion score of 4 points on a BASIC language pretest (see Appendix B) before any data was collected on them. This was an attempt to define the population at issue here as BASIC programmers with an intermediate to advanced knowledge of the language. These subjects were, of course, debriefed and awarded any reward that was due them. Each also read and signed an informed consent statement before participation.

Materials

The selection quiz. The selection quiz is included in Appendix B. It is made up of both background questions and questions about BASIC language syntax. A score of 4 out of a possible 9 was the criterion for inclusion in this study. All participants who contributed data attained this score.

The listings. Fifteen program listings were constructed by bugging the same BASIC program with three different bug types (Flow, Misuse, and Syntax) and then providing five different sorts of error messages for each of these bugs (see the appendices). The BASIC listing was run on a TRS-80 Color Computer, a Commodore 64, and a VIC-20 to be sure that the code used was of a fairly universal BASIC dialect. Each bugged version was then run on all three aforementioned computers to make sure all three machines reported the same sort of error occurring in the same line of the program. All bugs introduced into the program were restricted to a change in a single program line.

The first type of error was a simple syntax error constructed by misspelling a randomly selected keyword within the program. The keyword TO was misspelled TU in one line of the program. This same listing was used throughout the study. Data values within the program could also have been mistyped, but as this could also be considered an error in program solving strategy (Flow error) this was avoided.

The second kind of error (Misuse) is an error made in changing semantics (program flow) into syntax. Here the error is not in typing but in understanding the statement used. The general flow of the program is correct, as is the typing of the user. I used the FORTRAN abbreviation of "LT" for less than in place of the less than symbol (<) which is legal in BASIC to simulate this error.

The last kind of error (called Flow in Figure 2) is an error in overall problem solution. This error includes not anticipating data values, missing code, or any other error in the plan to solve the problem at hand. I constructed an error of this type by neglecting to dimension a variable and feeding the program too much data for the array.

Each of these errors were then flagged by five different error messages. Each subject saw all three types of errors but a single sort of error message. The first error message type was constructed by merely listing just the description of the error without a line number at the bottom of the program listing. This description was the error label the tested machines' BASIC interpreters presented when the bugged program was run on them (see Appendix C). The second type of error message involved the addition of the number of the line that was causing the error to the general error message given by the tested machines (e.g. Appendix D).

A third variety of error message involved giving text about the type of error involved and the BASIC statement the error occurred in (see Appendix F). The text used here was very closely based upon the TRS-80 Color Computer BASIC manual, "Getting Started With Color BASIC" (Smith & Yankel, 1981). Attention was also paid to The Commodore 64 Programmer's Reference Guide (Commodore Business Machines, 1982) and The Applesoft BASIC Programming Reference Manual (Apple Computer, Inc., 1978). Any major topics addressed in these other reference books not included in the TRS-80 text on an error message or BASIC statement was then worked into the appropriate error message.

A fourth type of error message included an indication of where the computer stopped translating the code on the line in error (e.g. Appendix E). These messages were produced by taking the actual location the TRS-80's BASIC stopped when the error message was generated by the BASIC interpreter.

The last type of error message involved a dump of the variables at the time the error message was generated and a listing of the last twenty lines run by the computer prior to the detection of the error condition (see Appendix G). As before, these diagnostics were actually produced by the test computers by simply using the program TRACE function and printing out the variables when the program "blew up".

In all of these messages, a deliberate effort was made to reduce the artificiality of the messages presented to the subject and to use diagnostics generated by a computer. This would help answer the question of whether error messages similar to those used here actually could be incorporated into an operating system.

<u>The Reference Card.</u> A BASIC reference card was also given to each subject. It is reproduced in Appendix A. This was based on the reference cards of several micro computers -- removing the graphics commands which are more specific to an individual machine. No other reference materials were allowed during the subject's task.

<u>Procedure</u>

Each subject was first given a copy of the BASIC language quiz to complete. Ten minutes were allowed for this which proved to be ample time to consider the test as one of power rather than one of speed. The test was then scored privately. If the criterion score was not reached, the subject was then dismissed at this point after debriefing and awarding them any compensation due.

Because it was necessary for the experimenter to indicate whether the program was debugged correctly during the session, the experimenter sat behind a wooden screen across from the subject throughout. This divider was intended to reduce the subject's anxiety as much as possible.

Directions were then read to the subject and three listings were presented one at a time along with the BASIC language summary card.

The order of presentation of the type of error was randomized for each subject to prevent practice from being confounded with the experimental effects.

The subject was then instructed to look at each listing and find the one error in it with the aid of the error message that followed the printed program listing. When the subject felt that he or she had found the error they were to then write the corrected line next to the one in error and present the listing to the experimenter. The directions emphasized that their performance was timed. The stop watch was stopped each time the listing was resubmitted and started again if the correct solution had not been reached. The subject was allowed 20 minutes to find and correct each error. Once a listing was debugged, or time had run out, it was collected before the next was presented.

Each subject was given one of three sorts of feedback when he/she presented a listing with a proposed correction. One type of feedback was that the error had been satisfactorily corrected; his/her time was then recorded. A second sort was that the error message would have been eliminated, but that the correction given did not make the program work right. A final kind of feedback was that the error message would remain. In all but the first case the subject was urged to keep trying.

Results

When this experiment was designed it was intended that an analysis of covariance be used to analyze the time to debug data. This analysis appears in Table I and the cell means in Table II. The covariate is the pretest given to each subject that is designed to measure BASIC programming ability. As there is only one covariate score for each subject, the within-subject factor cannot be corrected by the analysis of covariance (Winer, 1971).

The pretest does seem to be a good covariate as it is correlated $\underline{r}(\underline{df}=38)=-.606$ ($\underline{p}<.0005$) with the total time to debug all three listings, and does seem to actually test programmer skill as the first two demographic questions ("How long has it been since you programmed?" and "How long is your longest program?") were combined and correlated $\underline{r}(\underline{df} = 36)=.531$ with the rest of the test ($\underline{p}<.0005$).

Unfortunately, the distributions here appear to depart from normality. Hence two Chi-square statistics were computed to compare with the analysis of covariance. For the error type variable (the within subjects factor) a Friedman analysis of variance with ranked data for related samples was computed. The effect was at least as large as the

Table I
Analysis of Variance and Covariance on Raw Data

Summary Table

SOURCE	df	SS	MS	F
Err Messages	4	701285.44	165321.363	1.12
Sub. W/GRP	35	5488555.61	156815.875	
Err Types	2	7967470.62	3983735.310	27.81**
Interaction	8	840393.80	105049.225	.733
Error-w	70	10026334.90	143233.356	
Er. Mess(adj)	4	162679.295	40669.8238	2.793*
Sub. W/GRP(adj)	34	495028.145	14559.6513	

*$p < .05$, **$p < .01$

Table II
Cell Means for Raw Data

	error types			collapsed average
error messages	syntax	misuse	flow	
error type only	280.5	767,63	585.0	544.375
error type + line #	35.38	861.125	354.25	416.917
error type +# +place stopped	34.0	646.25	238.875	306.375
error type +# +text	53.13	651.75	491.375	398.75
error type +# +dump & trace	34.13	646.88	638.0	439.667
collapsed average	87.43	714.73	461.5	

analysis of covariance indicated: (2,_N_ = 40)=45.096 _p_<.001. A multiple comparison technique (Daniel, 1978) was applied and all of the group means were found to be significantly different from each other at the _p_<.05 level.

Then, using a Kruskal-Wallis one way analysis of variance by ranks a nonsignificant Chi-square was likewise obtained for the between-subjects factor of error messages: (4,_N_ = 40)= 5.214 _p_>.05. The Spearman correlations on the same data mentioned above were also very close to the Pearson values.

Because these non-normal distributions also exhibited somewhat heterogeneous variances a lograrithmic transformation was performed on the data. Also, the data in the misuse error type condition was truncated at the high end with a number of subjects not finishing the task in the required 20 minutes. To correct for this, an analysis of covariance was performed on the transformed data with the misuse condition data eliminated (see Table III & IV). The results also confirm the original analysis of covariance with the error type variable showing strong effects but with the error message variable making weak statisical significance and no interaction.

Taking these transformed data the four error message conditions with a line number were pooled (see Table V & VI)

Table III
Analysis of Variance and Covariance on Log Transformed Data Including Only Syntax and Flow Error Data

Summary Table

SOURCE	df	SS	MS	F
Err Messages	4	15.395	3.847	3.742*
Sub. W/GRP	35	35.999	1.029	
Err Types	1	68.730	68.730	67.196**
Interaction	4	6.719	1.680	1.642
Error -W	35	35.799	1.023	
Er. Mess(adj)	4	8.640	2.160	5.760**
Sub. W/GRP(adj)	34	12.750	.380	

*_p_< .05, **_p_< .01

Table IV
Table of Cell Means for Log Transformed Data Including Only Syntax and Flow Error Data

	error types		collapsed
error messages	syntax	flow	average
error type only	4.90	5.79	5.34
error type + line #	3.23	5.32	4.27
error type +# +place stopped	3.26	4.89	4.08
error type +# +text	3.56	5.60	4.58
error type +# +dump & trace	3.46	6.09	4.77
collapsed average	3.68	5.54	

Table V
Analysis of Variance on Log Transformed Data Syntax and Flow Data Line Number Versus No Line Number

Summary Table

SOURCE	df	SS	MS	F
Err Messages	1	10.81	10.81	10.11**
Sub. W/GRP	38	40.59	1.07	
Err Types	1	28.49	28.49	28.59**
Interaction	1	4.66	4.66	4.67*
Error-W	38	37.86	.996	

*$p < .05$, **$p < .01$

Table VI

<u>Table of Cell Means for Log Transformed Data, Syntax and Flow Error Data, Line Number Versus No Line Number</u>

	error types		collapsed
error messages	syntax	flow	average
error type only	4.90	5.79	5.34
error type + line #	3.37	5.42	4.42
collapsed average	4.14	5.63	

to produce a line number versus no line number analysis which revealed a stronger interaction effect _F_(1, 38) = 4.673, _p_<.05, which stemmed from a decrease in debug time with the help of a line number for the syntax errors (simple main effect _F_ (1,38) = 13.87, _p_<.01), but a lesser effect on debug time for flow errors when line numbers were provided (simple main effect _F_ (1,38) = 6.11, _p_<.05).

Discussion

Based upon all of the analyses presented, the largest effect is attributable to the different error types. Both the parametric and nonparametric statistics seem to strongly indicate this. Looking at Table II, the error type means will be found to be 87.425 sec., 461.5 sec. and 714.725 sec. for the syntax error, the flow error and the misuse error in that order. According to the Shneiderman model, the syntax error should require the least time as it does, but the ordering of the other two error types is reversed according to the theory. Though, as was pointed out earlier, only the syntax error and "something else" have been identified in the literature. I believe the most likely explanation for this reversal is that the example of the misuse error that I chose interacted with the interpreter that was the basis for the error messages and produced a very misleading error message. The computer reported that the error was one of allowing a subscript to go out of range while the code that was actually in error sat quietly, apparently working. Through discussions with professional programmers I have found that this "computer getting lost" phenomena is not all that rare. This is certainly at least another variable that must be worked into the theory and does seem to suggest error messages may play an important role -- though not always a helpful one. Of course, finding that there are other sorts of errors besides just syntax errors and "everything else" is important by itself.

The other variable, error messages, seem to have a weak effect that is obscured if the ability of the programmer is not controlled for. With the non-normality and truncation of some of the data here, this result, based solely on parametric statistics, should be interpreted with caution.

Most of the difference in error message groups lies between the no line number group and all of the other error messages. An outgrowth of this, the interaction in the second transformed analysis (Table V) is interesting from a theoretical standpoint with just knowing the line number helping with the debug of the syntax error while having a lesser effect upon the flow error. This is with the four different messages containing line number information lumped into one group. The line number alone was enough information to find a simple typing error but was less useful when repairing an error in the problem solving strategy of the program. The flow error condition still had some truncation; however, so this result should be considered only suggestive.

Two main difficulties appeared in doing the analysis of the data collected here, but I believe they both say something about this task and perhaps about computer programming in general. First, the distributions all had a bit of a bimodal shape to them, suggesting an "ah-ha" realization for some people and a "stumped" effect for the rest with very few between the extremes. Perhaps, then, either a programmer sees the problem right away or may go quite a time before doing so. Is this the immediate solution some programmers have reported whenever someone new looks at a listing that the first programmer cannot find the error in? Is this true of programming in general? Does this suggest that programmer interaction in a data processing department is very important?

The second difficulty was the effect of programmer ability. There are several indirect indications that programmer ability is an important variable in this task. First, without the analysis of covariance adjusting using the pretest (a test designed to measure programming ability), the error message effect is much weakened. The strong correlation between the time to find the bug and the pretest score seem to speak for programming ability having quite an effect upon time to find a bug. This was found despite a strong attempt to test only groups with an appreciable BASIC programming background. This seems to suggest that the ability of the programmer is a very important factor indeed and should be considered carefully in future research. A topic for future research might concern to what extent programming ability can make up for debugging aids like error messages.

References

Apple Computer Inc. (1978). _Basic programming reference manual_. Cupertino, CA: Apple Computer.

Arblaster, A. (1982). Human factors in the design and use of computing languages. _International Journal of Man-Machine Studies_, _17_, 211-224.

Baron, S., & Levison, W. H. (1977). Display Analysis with the optimal control model of the human operator. _Human Factors_, _19_, 437-457.

Bevan, N. (1981). Is there an optimum speed for presenting text on a VDU? _International Journal of Man-Machine Studies_, _14_, 59-76.

Boies, S. J. (1974). An interactive computer system. _IBM System Journal_, 2-15.

Bourne, Jr., L. E., Dominowski, R. L., & Loftus, E. F. (1979). _Cognitive Processes_. Englewood Cliffs, NJ: Prentice-Hall.

Commodore Business Machines, Inc. (1983). _Programmer's reference guide_. Wayne, PA: Commodore Business Machines.

Daniel, W. W. (1978). _Applied Nonparametric statistics_. Boston: Houghton Mifflin.

Davis, R. (1983). User error or computer error? Observations on a statistics package. _International Journal of Man-Machine Studies_, _19_, 359-376.

Ehrenreich, S. L. (1981). Query languages: Design recommendations derived from human factors literature. _Human Factors_, _23_, 709-725.

Fischer, C. N., and Mauney, J. (1980). On the role of error productions in syntactic error correction. _Computer Languages_, _5_, 131-139.

Gade, P. A., Fields, A. F., Maisano, R. E., & Marshall, C. F. (1981). Data entry performance as a function of method and instructional strategy. _Human Factors_, _23_, 199-210.

Geiselman, R. E., & Samet, M. G. (1982). Personalized versus fixed formats for computer-displayed intelligence messages. _IEEE Transactions on Systems, Man, and Cybernetics_, _SMC-12_, 490-495.

Gleitman, H. (1981). _Psychology_. New York, NY: W. W. Norton.

Gould, J. D., and Drongowski, P. (1974). An exploratory study of computer program debugging. _Human Factors_, _16_, 258-277.

Hansen, J. V. (1976). Man-machine communication: An experimental analysis of heuristic problem-solving under on-line and batch-processing conditions. _IEEE Transactions on Systems, Man, and Cybernetics_, _SMC-6_, 746-752.

Helander, G. A. (1981). Usability for business professionals. _IBM System Journal_, _20_, 20-40.

Helpler, S. P. (1976). Continuous versus intermittent display of information. _Human Factors_, _18_, 183-188.

Huchingson, R. D. (1981). _New horizons for human factors in design_. New York, NY: McGraw-Hill.

Huchingson, R. D., Williams, R. D., Reid, T. G., & Dudek C. L. (1981). Formatting, message load, sequencing method, and presentation rate for computer-generated displays. _Human Factors_, _23_, 551-559.

Jacob, J. K., Egeth, H. E., & Bevan, W. (1976). The face as a data display. _Human Factors_, _18_, 189-200.

Janelle, A., & Polis, M. P. (1980). Interactive hybrid computer design of a signaling system for a metro network. _IEEE Transactions on Systems, Man, and Cybernetics_, _SMC-10_, 555-570.

Jutila, S. T., & Baram, G. (1971). A user-oriented evaluation of a time-shared computer system. _IEEE Transactions on Systems, Man, and Cybernetics_, _SMC-1_, 344-349.

Lang, T., Lang, K., & Auld, R. (1981). A longitudinal study of computer-user behaviour in a batch environment. _International Journal of Man-Machine Studies_, _14_, 251-268.

Lucas, R. W. (1977). A study of patients' attitudes toward computer interrogation. International Journal of Man-Machine Studies, 9, 69-86.

Maguire, M. (1982). An evaluation of published recommendations on the design of man-computer dialogues. International Journal of Man-Machine Studies, 16, 237-261.

McDonald, N., & McNally, J. P. (1982). Query language feature analysis by usability. Computer Languages, 7, 103-124.

Moher, T., & Schneider, G. M. (1982). Methodology and experimental research in software engineering. International Journal of Man-Machine Studies, 16, 65-87.

Nickerson, R. S., Elkind, J. I., and Carbonell, J. R. (1968). Human factors and the design of time sharing computer systems. Human Factors, 10, 127-134.

Oberquelle, H., Kupka, I., & Maass, S. (1983). A view of human-machine communication and co-operation. International Journal of Man-Machine Studies, 19, 309-333.

Peace, D. M. S., & Easterby, R. S. (1973). The evaluation of user interaction with computer-based management information systems. Human Factors, 15, 163-177.

Pollack, M. (1976). Interactive models in operations reserach -- an introduction and some future research directions. _Computers and Operations Research_, _3_, 305-312.

Reisner, P. (1977). Use of psychological experimentation as an aid to development of a query language. _IEEE Transactions on Software Engineering_, _SE-3_, 218-229.

Ripley, G. D., & Druseikis, F. C. (1978). A statistical analysis of syntax errors. _Computer Languages_, _3_, 227-240.

Rouse, W. B. (1977). Human-computer interaction in multitask situations. _IEEE Transactions on Systems, Man, and Cybernetics_, _SMC-7_, 384-392.

Rouse, W. B., & Rouse, S. H. (1983). Analysis and classification of human error. _IEEE Transactions on Systems, Man, and Cybernetics_, _SMC-13_, 539-549.

Seamon, J. G. (1980). _Memory and cognition: An introduction_. New York, NY: Oxford.

Shneiderman, B. (1980). _Software Psychology_. Cambridge, MA: Winthrop.

Singer, A., Ledgard, H., & Hueras, J. F. (1981). The annotated assistant: A step towards human engineering. _IEEE Transactions on Software Engineering_, _SE-7_, 353-374.

Smith, D. G., & Yankel, L. (1981) _Getting started with color BASIC_. Fort Worth, TX: Tandy.

Sondheimer, N. K., & Relles, N. (1982). Human factors and user assistance in interactive computing systems: An introduction. _IEEE Transactions on Systems, Man, and Cybernetics_, _SMC-12_, 102-107.

Tai, K. (1978). Syntactic error correction in programming languages. _IEEE Transactions on Software Engineering_, _SE-4_, 414-425.

Treu, S. (1982). Uniformity in user-computer interaction languages: A compromise solution. _International Journal of Man-Machine Studies_, _16_, 183-210.

Wexley, K. N. & Yukl, G. A. (1977). _Organizational behavior and personnel psychology_. Homewood, IL: Richard D. Irwin.

Winer, B. J. (1971). _Statistical principles in experimental design_. New York: McGraw-Hill.

Zloof, M. M. (1982). A business language that unifies data and electronic mail. _IBM Systems Journal,_ ,_21_, 272-304.

Appendix A

BASIC language summary

BASIC language summary

WORD	PURPOSE	EXAMPLE
ABS	computes absolute value	Y=ABS(5)
ASC	returns ASCII code	A=ASC(T$)
ATN	returns arctangent in radians	Y=ATN(2)
CHR$	returns character of ASCII	P$=CHR$(T)
COS	returns the cosine of angle	Y=COS(7)
DATA	stores data--use READ to assign	DATA 12,14
DEF FN	defines numeric function	DEFFN(X)=X*3
DIM	dimensions arrays	DIM R(65),W(40)
END	ends program	END
EXP	returns exponential of number	Y=EXP(7)
FOR...TO STEP/NEXT	creates a loop	FOR L=1 TO 10: PRINTX:NEXT L
GOSUB	sends computer to a subroutine	GOSUB 800
GOTO	sends computer to a line number	GOTO 300
IF...THEN	performs a test	IF A=5 THEN 300
INT	converts number to integer	X=INT(5.2)
LEFT$	returns left portion of string	P$=LEFT$(M$,7)
LEN	returns the length of a string	X=LEN(SE$)
LET	assigns value to a variable	LET A$="JOBA"
LOG	returns natural logarithm	Y=LOG(353)
MID$	returns substring of a string	F$=MID$(A$,3,2)
ON...GOSUB	multi-way branch to subroutine	ON Y GOSUB 5,10
ON...GOTO	multi-way branch to line nos.	ON Y GOTO 19,20
PEEK	returns memory loc. contents	A=PEEK(32076)
POKE	puts a value into a mem. loc.	POKE 15872,255
PRINT	displays a number or message	PRINT "HI"
PRINT TAB	moves printing to column pos.	PRINT TAB(5)X
READ	assigns next item in DATA statement to a variable	READ A$,B$
REM	allows insertion of comment	REM IGNORED
RESTORE	sets pointer to first DATA item	RESTORE
RETURN	returns computer from subrout.	RETURN
RIGHT$	returns right part of string	Z$=RIGHT$(A$,5)
SGN	returns the sign of a number	X=SGN(A*B)
SIN	returns sin of angle	Y=SIN(5)
STOP	stops program execution	STOP
SQR	returns square root of a number	Y=SQR(5+3)
STR$	converts numeric to a string	S$=STR$(X)
TAN	returns tangent of angle	Y=TAN(45.7)
VAL	converts a string to a number	A=VAL(B$)

Appendix B

BASIC language pretest

```
Circle your one answer to the below questions

1) How long is the longest BASIC program you have written:
 -1 a)I have not written any programs
    b)less than 10 lines
    c)between 11 and 20 lines
    d)between 21 and 50 lines
 +1 e)more than 50 lines
2) How long has it been since you wrote in BASIC:
 +1 a)less than a week
    b)less than a month but more than a week
    c)a month
 -1 d)a couple of months or more
3) What would appear on the screen when this BASIC program
     is RUN:    10 READ C,A,B
                20 IF A<>2 THEN A=3
                30 B=C*A
  (+1 for       40 PRINT B
   answer C)    50 END
                60 DATA 1,2,3
    a)1, b) 6, c)2, d)3, e)0
4) What would happen when this program is RUN?
                10 GOSUB 30
                20 RETURN
                30 X=X+1
                40 PRINT X
                50 RETURN
                60 END
    a)the computer would print an error message
    b)the computer would print a "1"
    c)nothing would happen
 +1 d)the computer would print a "1" then an error message
5) What would appear on the screen when the following line
    is RUN:     10 REM PRINT "HI"
    a)HI                  +1 d)nothing
    b)0                      e)PRINT HI
    c)an error message
6) What does the STOP statement do?
    a)clear variables
    b)stops text from going to the printer
 +1 c)stops program execution
    d)turns off a TRACE command
    e)it is illegal--an error would occur
7) Which of the following lines would print "TOO BIG" only
    if A is larger than 5?
    a)IF A<5 THEN PRINT "TOO BIG"
 +1 b)IF A>5 THEN PRINT "TOO BIG"
    c)IF A<>5 THEN PRINT "TOO BIG"
    d)IF A THEN PRINT "TOO BIG"
    e)FOR A=1 TO 5:PRINT "TOO BIG":NEXT A
```

```
8)What does this statement do?
               10 DIM S(5)
 +1 a)dimensions the array S to five elements
    b)sets S=5
    c)divides S by 5
    d)adds 5 to S
    e)produces an error as it is not a BASIC statement
9)What COULD the following statement PRINT on the screen
    when RUN?
               100 PRINT X$;X
 +1 a)HELLO 5          d)nothing
    b)5HELLO           e)HELLO
    c)HELLO HELLO
```

Appendix C

BASIC program listing

Syntax error

Type only message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TU N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT


?SYNTAX ERROR
```

Appendix D

BASIC program listing

Syntax error

Type and line number message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM*********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TU N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT


?SYNTAX ERROR IN 260
```

Appendix E

BASIC program listing

Syntax error

Type, line number, and place message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TU N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT


?SYNTAX ERROR IN 260

LINE WAS: 260 FOR X=1 TU N
                      X
X MARKS WHERE COMPUTER STOPPED ON LINE
```

Appendix F

BASIC program listing

Syntax error

Type, line number, and text message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TU N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT
```

```
?SYNTAX ERROR IN 260
```

SYNTAX ERROR. This could result from a misspelled command, incorrect punctuation, open parentheses, or an illegal character.

FOR STATEMENT. Creates a loop in your program which the computer must repeat from the first to the last number you specify.
examples of this statement:

```
FOR X=2 TO 5
FOR A=1 to 10 STEP 5
```

Appendix G

BASIC program listing

Syntax error

Type, line number, dump and trace message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TU N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT
```

```
?SYNTAX ERROR IN 260

LAST 20 LINES RUN:
-170- -180- -190- -170- -180- -190- -170- -180- -190- -200-
-210- -220- -210- -220- -210- -220- -230- -240- -250- -260-

VARIABLES WHEN PROGRAM STOPPED WERE:
N=3
D(1)=1
D(2)=2
D(3)=3
F(1)=0
F(2)=0
F(3)=0
T=6
X=1
M=2
```

Appendix H

BASIC program listing

Misuse error

Type only message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM*********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO LT N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO
370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
```

```
500 REM DISPLAY STATISTICS.
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT


?BAD SUBSCRIPT ERROR
```

Appendix I

BASIC program listing

Misuse error

Type and line number message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM*********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO LT N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO
370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
```

```
500 REM DISPLAY STATISTICS.
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT


?BAD SUBSCRIPT ERROR IN 370
```

Appendix J

BASIC program listing

Misuse error

Type, line number and place message

```
10 REM *********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM*********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO LT N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO
370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
```

```
500 REM DISPLAY STATISTICS.
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT
```

```
?BAD SUBSCRIPT ERROR IN 370

LINE WAS: 370 IF CO LT N+1 THEN IF D(X)=D(CO) THEN IX=IX+1:
                                               X
CO=CO+1:GOTO 370

X MARKS WHERE COMPUTER STOPPED ON LINE
```

Appendix K

BASIC program listing

Misuse error

Type, line number and text message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO LT N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO
370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
```

```
500 REM DISPLAY STATISTICS.
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT
```

```
?BAD SUBSCRIPT ERROR IN 370
```

BAD SUBSCRIPT ERROR. The program was trying
to reference an element of an array whose
number is outside of the range specified in the DIM statement.

IF STATEMENT. Tests the relationship. If it
is true, the comptuer executes the instruction
following THEN. If it is not true the computer
executes the next line of the program.
examples of this statement:

```
IF A=5 THEN 300
IF B=0 THEN PRINT 0
IF A=3 THEN PRINT 4
```

Appendix L

BASIC program listing

Misuse error

Type, line number, dump and trace message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM*********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM D(N), F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO LT N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO
370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
```

```
500 REM DISPLAY STATISTICS.
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT
```

```
?BAD SUBSCRIPT ERROR IN 370

LAST 20 LINES RUN
-300- -290- -300- -310- -320- -330- -340- -350- -360- -370-
-380- -390- -400- -360- -370- -380- -390- -400- -360- -370-

VARIABLES WHEN PROGRAM STOPPED WERE:
N=3
D(1)=1
D(2)=2
D(3)=3
F(1)=0
F(2)=0
F(3)=0
T=6
X=3
M=2
CO=4
W=0
```

Appendix M

BASIC program listing

Flow error

Type only message

```
10 REM ***********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT


?BAD SUBSCRIPT ERROR
```

Appendix N

BASIC program listing

Flow error

Type and line number message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT


?BAD SUBSCRIPT ERROR IN 180
```

Appendix O

BASIC program listing

Flow error

Type, line number, and place message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT



?BAD SUBSCRIPT ERROR IN 180

LINE WAS: 180 INPUT D(X)
                        X
X MARKS WHERE COMPUTER STOPPED ON LINE
```

Appendix P

BASIC program listing

Flow error

Type, line number and text message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM**********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT
```

```
?BAD SUBSCRIPT ERROR IN 180
```

BAD SUBSCRIPT ERROR. The program was trying
to reference an element of an array whose
number is outside the range specified
in the DIM statement

INPUT STATEMENT. Caused the computer to
stop and await input from the device you specify. If you do not
specify a device, the computer will await input from the
keyboard
examples of this statement:

```
INPUT X$
INPUT X
```

Appendix Q

BASIC program listing

Flow error

Type, line number, dump and trace message

```
10 REM **********************************
20 REM Purpose of this program:
30 REM To compute Mean (the average of a
40 REM   set of numbers), Mode (the most
50 REM   often occuring score), Standard
60 REM   deviation (average amt. a set of
70 REM   numbers varies), and the RANGE
80 REM   of the set of numbers input by
90 REM   the user.
100 REM*********************************
110 PRINT "DESCRIPTIVE STATISTICS"
120 PRINT "BY DAVID LIONELL DAWSON"
130 REM KEYBOARD DATA ENTRY.
140 INPUT "NUMBER OF CASES";N
150 DIM F(N)
160 FOR X=1 TO N
170 PRINT "ENTER CASE #";X;
180 INPUT D(X)
190 NEXT X
200 FOR X=1 TO N
210 T=T+D(X)
220 NEXT X
230 M=T/N
240 PRINT "COMPUTING ... PLEASE WAIT ...";
250 REM SORT ROUTINE.
260 FOR X=1 TO N
270 PRINT "...";
280 FOR Y=1 TO N
290 IF D(X) < D(Y) THEN W=D(Y):D(Y)=D(X):D(X)=W
300 NEXT Y
310 NEXT X
320 REM CALCULATE MODE.
330 CO=1
340 W=0
350 FOR X=1 TO N
360 CO=X+1
370 IF CO < N+1 THEN IF D(X)=D(CO) THEN IW=IW+1:CO=CO+1:GOTO 370
380 IF IW>W THEN W=IW:MO=D(X)
390 IW=0
400 NEXT X
410 REM CALCULATE STANDARD DEVIATION.
420 T=0
430 PRINT"***";
440 FOR X=1 TO N
450 T=T+(D(X)-M)^2
460 NEXT X
470 SD=SQR(T/N)
480 REM CALCULATE RANGE.
490 RA=D(N)-D(1)
500 REM DISPLAY STATISTICS.
```

```
510 PRINT
520 PRINT"SUMMARY STATISTICS"
530 PRINT"CENTRAL TENDENCY"
540 PRINT"MEAN";M
550 PRINT"MODE";MO
560 PRINT
570 PRINT "VARIABILITY"
580 PRINT"RANGE";RA
590 PRINT"STANDARD DEVIATION";SD
600 PRINT
```

```
?BAD SUBSCRIPT ERROR IN 180

LAST 20 LINES RUN:
-170- -180- -190- -170- -180- -190- -170- -180- -190- -170-
-180- -190- -170- -180- -190- -170- -180- -190- -170- -180-

VARIABLES WHEN PROGRAM STOPPED WERE:
N=11
D(1)=1
D(2)=2
D(3)=3
D(4)=5
D(5)=87
D(6)=3
D(7)=6
D(8)=8
D(9)=2
D(10)=1
F(1)=0
F(2)=0
F(3)=0
F(4)=0
F(5)=0
F(6)=0
F(7)=0
F(8)=0
F(9)=0
F(10)=0
F(11)=0
X=11
```