EXPERIMENTAL INVESTIGATIONS
OF THE UTILITY OF FLOWCHARTS IN PROGRAMMING

Ben Shneiderman

Richard Mayer*

Don McKay

Peter Heller

Computer Science Department
Indiana University
Bloomington, Indiana 47401

*Department of Psychology
Indiana University
Bloomington, Indiana 47401

TECHNICAL REPORT NO. 36

EXPERIMENTAL INVESTIGATIONS
OF THE UTILITY OF FLOWCHARTS IN PROGRAMMING

BEN SHNEIDERMAN
RICHARD MAYER
DON MCKAY
PETER HELLER

AUGUST, 1975

Experimental Investigations

Of the Utility of Flowcharts in Programming

Ben Shneiderman
Richard Mayer*
Don McKay
Peter Heller

Computer Science Department
Indiana University
Bloomington, Indiana 47401

*Department of Psychology
Indiana University
Bloomington, Indiana 47401

Abstract

This paper describes previous research on flowcharts and a series of controlled experiments to test the utility of detailed flowcharts as an aid to program composition, comprehension, debugging and modification. Our results showed no statistically significant difference between flowchart and no flowchart groups, thereby calling into question the utility of flowcharting. A program of further research is suggested.

Keywords

flowcharts, program composition, program comprehension, debugging, modification, experimental testing, human factors.

CR Categories

1.5, 4.0

Introduction

Flowcharts have been a part of computer programming since the introduction of computers in the 1940's. Goldstein and von Neumann (1947) presented a system of describing processes using operation, assertion, and alternative boxes. Since that time, flowcharting has become an accepted part of the computer programming field. The integration of flowcharts was so complete that a national standard was proposed in 1963. However, the development of more powerful programming language necessitated revisions in the original flowcharting schemas. For example, the FORTRAN DO statement has caused many textbook authors and programmers to create their own set of conventions for this construct. Other flowcharting schemes have developed including "structured flowcharts" (Nassi and Shneiderman,1973) and its variations (Chapin, 1974).

Goldstein and von Neumann felt that the "coding [process] begins with the drawing of flow diagram." Prior to the coding process, the algorithm had been identified and understood. The flowchart represented a high level definition of the solution to be implemented on a machine. Although they were working only with numerical algorithms, they proposed a programming methodology which has become standard practice.

Approximately a dozen texts are entirely dedicated to teaching flowcharting. Mario V. Farina, in his book Flowcharting (1970), expresses the opinion that flowcharting is an art requiring practice and that a flowchart should be developed before a program is coded. This opinion is practiced in many professional and educational institutions.

In *Flowcharting Techniques* (1971), Marilyn Bohl holds that flowcharting helps "distinguish between the procedure a computer program is written to express and the syntactical details of the language in which the program is written." She feels the flowchart is "an essential tool in problem solving" and states, "The person who cannot flowchart cannot anticipate a problem, analyze the problem, plan the solution, or solve the problem."

Programming language texts strikingly reflect the differing opinions about flowcharting. An examination of 45 FORTRAN texts showed that 14 of them employed flowcharts extensively and 19 texts used them occasionally. The remaining 12 used no flowcharts. Another teaching philosophy is seen in introductory "computer science" texts. Being language independent they teach only the principles of programming. In these texts flowcharts are the main vehicle with which ideas are expressed. This approach suggests that flowcharting and programming are independent tasks.

Flowcharting takes many forms of which the hand-drawn type is only one. The introduction of computer-"drawn" flowcharts produced from *completed* programs was intended to greatly aid a future programmer's comprehension of the program when attempting debugging or modification. However, the usefulness of such ex post facto flowcharts is hotly disputed. In *The Program Development Process* (1974), Joel D. Aron maintains that such flowcharts are useless to a programmer when diagnosing errors. In such cases "the most helpful data is the program listing itself." Concurring with this opinion is Gerald M. Weinberg (1971) who states, "We find no evidence that the original coding plus flow diagrams is any easier to understand than the original coding itself---except to the original programmer."

Frederick P. Brooks (1975) is especially vehement in his criticism of the flowchart as documentation, referring to it as "a curse," "a space-hogging exercise in drafting," and "a most thoroughly oversold piece of program documentation." Resolving the question of whether or not flowcharts are an aid in programming could significantly affect the manner in which programming is taught, documented, and practiced.

In a series of experiments with naive programmers, Mayer (1975) demonstrated that providing a conceptual hardware model of a computer and its operations during instruction, aided performance on postinstructional test items which required interpretation of programs. Other subjects who utilized a flowchart or a flowchart with the model performed well on test items which required composition of a program but not as well as the model group on "interpretation" items. This experiment indicated that the use of flowcharts may assist program composition but may hinder learning and performance on interpretation tasks.

The basic tasks of programming have been delineated as program composition, comprehension, debugging, and modification (Shneiderman, 1975; Weinberg, 1971). It is within the scope of these tasks that advocates of flowcharts have insisted that flowcharts provide a pictorial representation of the logic or flow of control of a program which is indispensable.

Flowcharts are not specific to computer programming. Considerable research and investigation of flowcharts as a tool for communication has been conducted in the human factors field. Lewis, Horabin, and Gane (1967) discussed the utility of an "algorithmic approach" to official rules and regulations. Their concept of an "algorithm" "extends the concept to cover any rule of thumb procedure, any recipe---not necessarily mathematical---for achieving a desired outcome." In most of their examples, the "algorithm" for a process was presented as a flowchart which they claimed was less susceptible to misinterpretation and less time consuming to execute than a prose description. Kammann (1975) displayed remarkable results for a telephone dialing task that required reference to a set of instructions. Housewives and Bell Telephone Laboratory professionals committed fewer errors and the housewives expended less time when using a flowchart diagramming the dialing process than when supplied with the normal prose description from a phone book. Kammann gave the following reasons for advocating the use of flowcharts for this type of problem:

- they move major decision criteria forward in the information sequence
- they reduce the complexity of the prose needed to describe the contingent relations
- they distinguish more clearly between relevant and irrelevant information for a given problem
- they reduce the amount of actual information to a reasonable load.

Wright and Reid (1973) reported similar success with flowcharts over prose descriptions for an unfamiliar "algorithm" of hypothetical mechanisms for space travel. However, when subjects were required to solve the space travel problems from memory, performance for a flowchart group decayed over time while the performance for a prose group increased. They suggested that the flowchart representation of the "algorithm" was encoded visually and subject to "becoming less distinct over successive trials." The prose was encoded verbally and it was possible for the subjects to continue memorizing the material after it was removed (Shiffrin, 1970; Tulving, 1962).

The human factors research, although showing interesting results for flowcharts, is not directly applicable to computer programming. First, programs are not expressed in vague prose but are algorithms represented in a precisely defined language. Second, previous research only measured performance for a task which required little understanding of the "algorithm" itself. Third, most of the flowcharts used were limited to one page. Finally, subjects were not given redundant information; they received either the flowchart or the prose, not both.

The goal of the research described in this paper was to determine the usefulness of detailed flowcharts in the computer programming tasks of composition, comprehension, debugging and modification.

Experiment I (Composition)

This first experiment was designed to study how the creation of a detailed flowchart assisted the subjects in composing a program. Much of the literature on flowcharting supports the contention that they are most helpful as a program design aid which helps to clarify the problem to the programmer.

METHOD

Subjects: The subjects were students in an introductory computer programming course using FORTRAN. The textbook used flowcharts to illustrate program development and flowcharts were used by the instructor. The experiment was conducted by including the materials as part of the second of three in-class examinations which constituted the major part of the course grade.

Procedure and Materials: Thirty-four subjects (flowchart group), selected randomly, received test instructions which indicated that they were to write a flowchart and then a program for a given problem. The flowchart counted for fifteen points, the program 25 points. Twenty-eight subjects (non-flowchart group) were instructed to merely write the program, which counted for the full forty points. The subjects were given as much time as they wanted to complete the test. The grading was done by a graduate student with much experience in grading programs and in consulting with students about programs. The results were returned to the students at the next meeting of the class.

RESULTS

The scores on the program composition task were normalized to 100 percent. The flowchart group mean score was 94 while the non-flowchart group mean was 95. A t-test showed no significant difference between the two groups. The flowchart group had a mean score of 13.1 on the flowchart or 87.3 out of 100.

DISCUSSION

The requirement to produce a flowchart seemed to have no benefit or harm on the subjects' ability to prepare a program described by that flowchart. This was inspite of the fact that the problem had been chosen to favor the flowchart group by having a relatively complicated branching pattern. The relatively good scores indicate that all the subjects found the task to be straightforward.

An earlier pilot study with the same design and procedures had produced similar results. In the pilot flowchart group two subjects achieved perfect scores and 14 out of twenty-three subjects scored fifty percent or above. In the pilot non-flowchart group four subjects achieved a perfect score and seventeen out of twenty-six subjects scored fifty percent or above. The pilot study was run earlier in the term with a simpler problem, but one which the subjects found more challenging, given their shallower background.

Experiment II (Comprehension)

A second often cited beneficial aspect of detailed flowcharts is as an aid to comprehension of programs. Flowchart proponents argue that if a detailed flowchart is studied in conjunction with a program, comprehension can be improved. This experiment was designed to test this hypothesis. Since Experiment I had not shown flowcharts to be useful for composition in a simple program with novices, this experiment drew on more complex program structures, but still with novice programmers as subjects.

METHOD

Subjects: The subjects were again students in an introductory computer programming course using FORTRAN and similar to the previous subjects. The experiment was conducted by including materials as part of the third of five in class examinations which constituted the major part of the course grade.

Procedure and Materials: Sixty of the one hundred points on the examination were related to the experiment. Two forms of the examination were prepared. The first form contained two programs (27 and 24 FORTRAN statements) with a flowchart for the first program only, while the second form contained the same two programs but a flowchart for the second program only. Twenty-five randomly selected subjects received the first form, 28 received the second form. The comprehension questions, which were the same on both forms of the test, required the subjects to determine the values printed for various inputs and to trace the flow of execution. The subjects were given as much time as they needed to complete the test. Grading was simplified since the answers were clearly correct or not correct.

The results were returned to the students at the next class meeting.

RESULTS

An analysis of variance for the scores indicated that the only significant result was that problem 2 was more difficult than problem 1. Subjects who had flowcharts did not perform differently than those who did not have flowcharts. The mean percent correct for each group and problem appear below as Data Table I.

Data Table I

Mean Percent Correct for Experiment II

	Problem	
	1	2
Flowchart on 1	94.4	89.6
Flowchart on 2	97.0	94.4

DISCUSSION

The availability of a flowchart neither benefitted nor harmed the subjects in the comprehension task. This was surprising since the programs had been designed with a large number of complex transfers of control; precisely the situation which is purported to be most advantageous to the flowchart groups. Informal observation during the exam showed that most subjects were rarely referring to the available flowchart but preferred to study the program directly.

Experiment III (Comprehension and Debugging)

This experiment, which used subjects from an intermediate experience level, measured the effect of flowcharts as opposed to no flowcharts in debugging and comprehension of programs.

METHOD

Subjects: The subjects selected were basically from one programming experience level. They were all in intermediate FORTRAN programming courses but were divided into two groups due to the flowcharting environment they were learning in. Group I consisted of 43 subjects who had had some previous instruction about flowcharts and presently were using a book employing flowcharts but were under no obligation to use or turn in flowcharts with their assigned programs. Group II consisted of 27 subjects who were required to write a flowchart before doing the actual programming and to hand it in with their programs. The data from each group was analyzed separately and later all data was combined and analyzed. All subjects knew the experiment would have no bearing on their course grade.

Design: The experimental design consisted of a control subgroup and two experimental subgroups for both Group I and Group II. The members of each subgroup were randomly selected, each subgroup containing approximately equal numbers of subjects. The control subgroups received no flowchart. The first experimental subgroups received a detailed four-page (micro) flowchart while the second experimental subgroups received a more general single-page (macro) flowchart.

Materials: All the subjects received the same compiled listing (with line numbers) and its corresponding output. Depending upon which subgroup (experimental factor) they were being tested in a subject would also receive either a micro flowchart, a macro flowchart, or no flowchart at all. The program used was a moderately difficult tic-tac-toe playing program written in FORTRAN. The program consisted of a main program which was 81 lines long and two subroutines. The main program was commented and had several DO loops as well as a controlling IF loop. The program had three non-syntactic bugs placed in it. It produced incorrect output which only reflected one of these bugs. Both the program and the flowcharts were taken from Sturgul & Merchant (1973).

Procedure: The experiment took the form of a three-part test. In the first part once the subjects had received a·program listing and one of the three flowchart possibilities each was given a combination instruction/answer sheet. In the instructions subjects were told:

(1) They had been given a listing of a tic-tac-toe playing program which was not supposed to lose.

(2) The program had at least one bug in it which was apparent in the output.

Next they were instructed to:

(1) Study the listing and output in conjunction with any flowchart they had received (or on its own if none was received).

(2) Find the bug(s) and explain how to repair them. (Bugs were reported by referring to the line number and writing the necessary revision.)

For this task the Group I subjects were given 40 minutes and the Group II subjects were allowed 50 minutes after which the answer sheets were collected. (The subjects retained the listings and flowcharts.) After a short break the second part of the test was administered.

In the second part the subjects were first informed of the bugs and told to make the corrections in their listings. Next they were instructed to answer 11 multiple-choice questions for which the Group I and II subjects were allotted 20 and 30 minutes respectively. The questions which were designed to measure program comprehension tested basic programming knowledge, hand simulation of the (corrected) program, and other problems necessitating knowledge of the program. (The subjects still had their listings and flowcharts for references while answering the questions.)

Finally, the subjects were asked to respond to questions concerning their feelings about how well they had done in a questionnaire. In each of the questionnaire questions the subjects responded on a scale of 0 to 9. All subjects were asked how well they thought they understood the program. Any subject who had received a flowchart was further asked to respond to questions concerning the usefulness of the flowchart in their understanding of the program and in finding the bugs.

RESULTS

The data gathered from the two separate subject groups is shown below. The two groups were from entirely separate subject pools tested under different conditions and therefore the magnitude of the scores cannot be compared between groups. However, the trends within each group can be compared.

The Group I data (those who do not normally use flowcharts) reflects their background with the non-flowchart subjects having the highest mean scores for both the debugging and comprehension tests, micro next, and macro last. The Group II data (those who do use flowcharts) reflects a quite different orientation. The micro flowchart subjects had the highest average scores for both tests in Group II, macro subjects had the next best scores, and the non-flowchart subjects came in last.

This would seem to indicate a correlation between a person's background with flowcharts and their usefulness in programming. However, t-tests between the three combinations of experimental conditions (none-macro, none-micro, and macro-micro) for the three test results (debug score, comprehension score, and total) for both Groups I and II, as well as an analysis of variance, showed none of the results to be statistically significant even at the 10% level.

As stated before one bug caused an obvious error in the output and this bug was the most easily found of the three. Nearly 70% of the subjects found and corrected it. In contrast, 12% of the subjects found two bugs and only three subjects out of 70 found all three bugs. Of these three, two also answered all the comprehension questions correctly. This suggests that the two tasks complement each other and that to successfully debug a program one must have a thorough comprehension of it. This trend holds fairly closely for those finding two bugs but breaks down for those finding only one or no bugs.

The comprehension test results parallel the debugging test for both Groups I and II. For both groups the flowchart subgroup which had the highest average debug score also had the highest mean for comprehension. An item analysis of the comprehension questions was inconclusive and no important results could be drawn from it. The subjects perceived very accurately how well they understood the program. To show this, the subjects were grouped by the number of questions they answered correctly. The average questionnaire response to how well they thought they understood the program was obtained for each level of "correctness." At each increment in the number of correctly answered questions the average response would generally increase suggesting that the subjects were in tune with their performances.

The questionnaire responses to how useful subjects found the flowchart in debugging yield an unexpected result. For both the macro and micro flowchart subjects in both subject groups those who found only one bug (of three) felt the flowchart helped most in debugging. Those who found 0, 2 or all three bugs rated the flowchart's usefulness at a lower level. Perhaps this was because the one bug which manifested itself in the output was traceable through the flowchart while the less obvious ones had to be found by hand simulation using the listing itself.

DISCUSSION

These results are strong evidence in favor of the contention that flowcharts do not seriously affect a programmer's ability to comprehend or debug a program. Depending on the programmer's background flowcharts may help or hinder performance, but the results show the difference is not at a significant level.

The trend of the scores in favor of flowcharts for those who use them (Group II) and away for those who do not (Group I), was expected. The result that the differences were not significant was surprising and warrants further experimentation. Especially valuable are experiments which put flowcharting and programming in "head to head" competition. One such experiment would be to assign a program having one group design a flowchart for it and another group write the program without a flowchart. Next, those who had written flowcharts would be given time to code up their programs. Then both groups would be timed and compared on how quickly they could get their programs running.

Another test of how useful the old adage, "flowchart before programming," would be to assign a program and have subjects design an initial flowchart which would be kept by the experimenter. The subject would then write the program, get it running (correctly), and finally make a flowchart of the completed program. The original and final flowcharts would then be compared for changes made in the original flowchart while coding.

The results of the questionnaire give some interesting insights into the subjects' thoughts about the flowcharts. Somewhat surprisingly, the answers given by both the Group I and II subjects are nearly identical in all respects. For both there was a pronounced positive correlation between the number of questions answered correctly and how well the subjects thought they understood the program in every experimental condition.

The macro flowchart was not of much use either to the Group I or Group II subjects. Subjects in the macro flowchart condition for Group I were hindered slightly while the Group II subjects were aided slightly. It would appear that it is more useful in finding major flow of control errors than slight logic errors. Interestingly, the detailed (micro) flowchart listed nearly all the statements from the program (as they should have been) so anyone who had directly compared the listing and flowchart would have found the flowchart very useful in debugging. But even the Group II (flowchart users) rated the micro flowchart unprofitable if they found two or more bugs.

Table II

Results for Experiment III

Average % Correct (Comprehension)

	Flowchart Used		
	None	Macro	Micro
Group I	52	34	46
II	53	55	76

Average % Correct (Debugging)

	Flowchart Used		
	None	Macro	Micro
Group I	12	11	4
II	29	26	45

Average % Correct (Total Score)

	Flowchart Used		
	None	Macro	Micro
Group I	34	23	27
II	42	42	62

Experiment IV (Modification)

The focus of this experiment was the use of flowcharts as a program documentation tool. The basic question was whether flowcharts can assist the modification of an existing program. Since it is claimed flowcharts describe the logic of a program, the task of discovering the placement of a modification should be easier when the programmer has a flowchart of the program. This experiment compared the performance of intermediate programmers in a modification task in which some subjects received flowcharts and others did not. Two levels of flowcharts were used: a detailed, statement by statement flowchart (micro flowchart) and a higher level flowchart (macro flowchart).

METHOD

Subjects: The subjects for this experiment were students of a second semester programming course at Indiana University and Purdue University. The experiment was conducted during a regularly scheduled class meeting. Subjects from Indiana University were 33 students who were not required to design flowcharts as part of their assignments. These subjects were exposed to flowcharts by the textbook used for the course. The 37 subjects from Purdue University who participated in the experiment were required to turn in flowcharts with their programming assignments.

Materials: The booklet given to each subject contained a set of instructions, a FORTRAN program, a loader map, sample output, a set of three modification descriptions, and a biographical questionnaire. The 78-line FORTRAN program produced semester grade reports for a fictitious college. The program had 48 lines of FORTRAN code and

27 lines of comments of which 23 appeared as a comment block at the beginning of the program. The output from the program was a set of semester grade reports for all student records input. There were three separate program listings for the subjects to indicate their modifications.

In addition to the booklet, one third of the subjects received a one-page macro flowchart, another third were supplied with a three-page micro flowchart, and the remainder were given no flowchart.

Procedure: The subjects were instructed to peruse the instructions, program and flowchart for approximately 5 minutes. Each subject was asked to make the modifications to the existing program according to the modification description provided. A total of three modifications were to be attempted in the 45-minute period allowed for the experiment. Subjects were told their modifications would be graded on correctness and runability. At Indiana University, each subject was timed individually by the experimenters for each modification. At Purdue, each subject was responsible for recording the amount of time spent on each modification.

RESULTS

Since unequal number of subjects were present in the experimental groups, the data obtained from 12 subjects was not included in the following analysis. Of these subjects, 5 did not finish and 7 had GPA's of 2.5 or below on a 4.0 scale, where 4.0 is an "A." The results reflect the data from 60 subjects, 30 from each university, and 10 in each of the 6 groups. Also, most subjects did not finish Modification III and it was not included in these results. At Indiana University, no one finished Modification III; 6 subjects from the micro flowchart group, 8 subjects from the macro group, and 7 subjects from the non-flowchart group started Modification III. In the Purdue groups, 1 subject from each of the groups finished Modification III. Of those who did not finish, 7 from each of the groups had started the problem.

Modifications I and II were graded and a percent correct determined for each subject. The actual grading was done by the experimenter who had spent the semester as an Associate Instructor for the Computer Science Department of Indiana University and who had graded programs perviously. The types of errors made were:

- incorrect formatting of output
- incorrect placement of modification within the existing program
- violation of the modification description
- violation of FORTRAN syntax
- errors of omission.

The mean percent correct for each group is displayed in Data Table III. An analysis of variance for the scores showed that the University factor (Indiana vs. Purdue), the Modification factor (Modification I vs. Modification II), and the interaction of University and Modification factors were all significant.

These results indicate that the Purdue groups made fewer errors, Modification II was more difficult than Modification I, and the Purdue groups performed better than the Indiana groups on both modifications.

Data Table III

Mean Percent Correct

Univ.	Flowchart Aid	Mod I	Mod II
Purdue	None	73	85
	Macro	88	77
	Micro	87	81
Indiana	None	77	64
	Macro	77	71
	Micro	77	59

An analysis of the types of errors made showed that the second type of error listed, "incorrect placement of modification within the existing program," was as frequent in each of the 6 groups. The type of flowchart aid; none, macro, or micro, was not a significant factor in these results.

An analysis of variance for the time measure indicated that neither the type of flowchart nor any of the possible interactions was significant. The mean times are displayed in Data Table IV. While the means reflect that some groups did perform better than others, the variance in these groups was extremely large. The time measure did not turn out to be a useful measure.

Data Table IV

Mean Time per Modification in Minutes

Univ.	Flowchart Aid	Mod I	Mod II
Purdue	None	15.5	15.3
	Macro	16.1	14.2
	Micro	14.0	14.1
Indiana	None	15.6	16.7
	Macro	15.4	13.9
	Micro	16.0	17.9

DISCUSSION

The results from this experiment showed no advantage for groups receiving flowcharts in a program modification task for either time or percent correct measures. An analysis of the types of errors made indicated that most of them were errors which flowcharts may not decrease such as coding errors and errors of omission. After discarding these types of errors, no significant difference can be reported between the flowchart and non-flowchart groups for incorrect positioning of the modification within the existing program. The results from this experiment contrast with the human factors research in prose and "algorithmic" representations of rules and regulations in which flowcharts have been shown to aid comprehension. As stated previously, there were several task differences between the present experiment and the human factors research.

Lewis, Horabin, and Gane (1967) claimed that a flowchart description of an algorithm was less ambiguous than a prose description. A program description of an "algorithm" is also a well defined entity. Both programs and detailed flowcharts of the same "algorithm" can be presented in a confusing manner, but if a detailed flowchart and program displayed the same logic, they should be equivalent representations. Part of this experiment was designed to test the utility of detailed, micro flowcharts in a program modification task. The intermediate programmers who were given both the program and the micro flowchart of the program did not perform better in the modification task than those who received only the program. This indicated that a micro flowchart and a program do provide equivalent information for a modification task and that the micro flowchart does not provide any additional information. Several questions still remain about micro flowcharts. First, which is easier to understand, a program description of an "algorithm" or a micro flowchart. The research of Wright and Reid (1973), in which subjects were required to memorize an algorithm showed that a flowchart representation was more difficult to memorize than a prose description. While similar research does not exist for a programming language description, it might be reasonable to assume that in a memorization task similar results could be obtained. Another improtant area of concern is ease of execution. This experiment did not explicitly require the existing program to be executed or hand simulated. This question should be investigated further especially in conjunction with determining what role accurate algorithm execution plays in the programming tasks of composition, comprehension, debugging, and modification.

Another part of this experiment dealt with higher level or macro flowcharts. Again, the subjects who were given both a macro flowchart and a program did not perform differently from those who were supplied with just the program. This result is quite surprising. A macro flowchart is not equivalent to the program it describes since a one-to-one correspondence between the flowchart and the program does not exist. Kammann's (1975) description of the usefulness of flowcharts versus prose can be slightly modified to state what information an ideal macro flowchart conveys:

- they move major decision criteria forward
  in the information sequence
- they reduce the complexity of the code
  needed to describe the contingent relations
- they distinguish more clearly between relevant
  and irrelevant information for a given problem

- they reduce the total amount of information to be understood to a reasonable load.

In these terms, macro flowcharts should be an indispensable tool for any of the programming tasks. Such a flowchart would introduce the true logic of the program totally divorced from the implementation details and allow the programmer to view the program from a more general perspective. However, these ideas may be suggesting good programming practices which have already been identified. For program composition, they suggest clearly identifying the problem and the initial solution statement. The critical factor in program modification might be understanding the existing program at a macro level which allows identification of where to incorporate the modification and what effects the proposed modification will have on the remainder of the program. A similar statement could be made for program debugging. The macro flowchart or at least some general presentation of the program seems intuitively useful. Since this experiment did not show any significant results for the macro flowchart group, the question of the utility of macro flowcharts as a vehicle of expression remains confused. Perhaps even the macro flowchart is equivalent to the program it describes in the same manner as the micro flowchart and thus provides no additional information.

For both macro and micro flowchart groups, their failure to outperform the other group could have been attributable to several experimental variables. First, the program used in this experiment was not exceedingly long nor complex. It is possible that a replication of the current experiment with a significantly longer and more complex program may show different results. This may be especially true for the macro flowchart since it is viewed as an organizer which might facilitate the "chunking" of the program into logical modules. Results for the micro flowchart group in this proposed experiment would not be expected to differ from the non-flowchart group except for a task which required "algorithm" execution and little understanding of the process itself. Second, the topic of the program might have been familiar to both groups of college students, however, the Purdue groups were possibly not as familiar with the grading system since it was modeled after the Indiana University system. Finally, the program had a comment block at the beginning of the program which explained the general workings of the program and this may have been enough information on which to base a modification.

Experiment V (Comprehension)

The previous experiments reported here have not shown what information about an algorithm can be obtained from a flowchart representation. Experiment IV argued that detailed flowcharts and program representations provide equivalent information. Also, since most subjects scored well in the first comprehension experiment (Experiment II), a more difficult problem was designed for Experiment V. Experiment V investigated the information content of detailed flowcharts in a comprehension task. Two types of comprehension items were presented: algorithm (low level) execution problems and interpretation (higher level) problems.

METHOD

Subjects: The 58 subjects for this experiment were students of an 8-week summer session introductory course in computer science at Indiana University. The experiment was conducted as the third quiz of the term at the beginning of the sixth week of the course. Subjects were familiar with FORTRAN and flowcharts.

Materials: Each subject received a quiz booklet. Three different booklets were used. Each contained an algorithm and a set of questions about the algorithm. The algorithm used was a two-way array merging algorithm. The subjects first exposure to the algorithm was the quiz. One group of subjects received a 23-line FORTRAN program of the algorithm, another group was given a one-page detailed flowchart of the algorithm and the last group was provided with both the detailed flowchart and the FORTRAN program. The quiz questions were of two types: execution problems and interpretation items.

The two execution items asked for the algorithm's output given a set of input values. The three interpretation questions were concerned with the operation and properties of the algorithm.

Procedure: Each subject received a quiz booklet and was given general instructions for the quiz. The subjects were given 25 minutes for the quiz.

RESULTS

The subjects' quizzes were graded on a 100-point scale with each problem worth 20 points. Seven subjects were not included in the results; three dropped the course and four had class averages below "D" level. The mean percent correct for each group and the type of question appear below as Data Table V. An analysis of variance indicated that all groups performed equally well. The means for each group suggested that the group which had only the program performed the best. Again, the variance within each group was large and the mean percent correct scores cannot be considered statistically different.

Data Table V

Mean Percent Correct for Experiment V

	Type of Question	
Group	Execution	Interpretation
Flowchart	48.5	51.2
Program + Flowchart	56.9	50.0
Program	57.8	62.4

DISCUSSION

The results of this experiment indicated, as argued in Experiment IV, that the type of information obtained from a detailed flowchart and a program appear to be equivalent. The most interesting result is the comparison of the program group versus the other two groups for the two types of questions. For the execution items the program group scored the highest and the flowchart group scored the lowest. This result, although not statistically reliable, suggested that a program may be more understandable than a flowchart for execution type problems. For the interpretation items, again the program group performed the best, suggesting the program representation may be supplying additional information which a flowchart cannot. The performance of the program group versus the program plus flowchart group suggested that the presentation of redundant information may hinder understanding necessary for a general understanding of an algorithm. In terms of the human factors research, this experiment was similar to Mayer (1975) which demonstrated different learning outcomes for flowchart groups and to Wright and Reid (1973) which showed different results for flowchart and prose descriptions that were memorized by subjects.

General Discussion

Although our original intention was to ascertain under which conditions detailed flowcharts were most helpful, our repeated negative results have led us to a more skeptical opinion of the utility of detailed flowcharts under modern programming conditions. We repeatedly selected problems and tried to create test conditions which would favor the flowchart groups, but found no statistically significant differences between the flowchart and non-flowchart groups. In some cases the mean scores for the non-flowchart groups even surpassed the means for the flowchart groups.

We conjecture that detailed flowcharts are merely a repetitious presentation of the information contained in the programming language statements. The flowcharts may even be at a disadvantage because they are not as complete (omitting declarations and input/output formats) and require many more pages than do the precise concise programming language statements.

The advent of top down design techniques and structured control structures reduce the utility of detailed flowcharts since programmers have learnt to think in higher level concepts than those represented by standard detailed flowcharts. This suggests that further experiments should be done with macro flowcharts and structured flowcharts (Nassi and Shneiderman, 1973).

Our results should be replicated with a wide variety of programmers and problems. Especially important are studies on large complex programs. Another important research direction would be to study professional programmers who feel that flowcharts are essential in their work. Is their dedication to this technique well-founded or would their time and energies be better spent in more careful program design or documentation. It has also been suggested that detailed flowcharts are more meaningful than programming language statements for managers and non-programmers. This possibility should also be investigated.

In summary, our experiments have not demonstrated the utility of flowcharts in program composition, comprehension, debugging, or modification. Further work is necessary to replicate the results to explore other areas where flowcharts may be helpful, and to study other forms of flowcharting.

Bibliography

1. American Standards Institute. Proposed american standard flowchart symbols for information processing. Comm. ACM 6, 10 (1963), 601-604.

2. Aron, J. The Program Development Process; The Individual Programmer, Addison-Wesley, Reading, MA (1974), 104-106.

3. Bohl, M. Flowcharting Techniques, Science Research Associates (1971), 53.

4. Brooks, F.P. The Mythical Man-Month, Addison-Wesley, Reading, MA (1975).

5. Chapin, N. New format for flowcharts. Software Practice and Experience 4, 4 (1974), 341-357.

6. Farina, F. Flowcharting, Prentice-Hall (1970), iii.

7. Goldstein, H.H., and von Neumann, J. Planning and coding problems for an electronic computing instrument, part II, vol. I. Report prepared for the US Army Ordinance Department (1947). Reprinted in J. von Neumann, Collected Works, A.H. Taub (Ed.), vol. V, McMillan, New York, 80-151.

8. Kamman, R. The comprehensibility of printed instructions and the flowchart alternative. Human Factors 17, 2 (1975), 183-191.

9. Lewis, B.N.; Horabin, I.S.; and Gane, C.P. Flowcharts, Logical Trees and Algorithms for Rules and Regulations, Her Majesty's Stationary Office, London (1967).

10. Mayer, R.E. Instructional variables in meaningful learning of computer programming. Indiana Mathematical Psychology Program Report Series No. 75-1, Indiana University (1975).

11. Nassi, I., and Shneiderman, B. Flowcharting techniques for structured programming. SIGPLAN Notices 8, 8 (1973), 12-26.

12. Shiffrin, R.M. Memory search. In Models of Human Memory, D.A. Norman (Ed.), Academic Press (1970).

13. Shneiderman, B. Experimental testing in programming languages, stylistic considerations and design techniques. Proc. NCC 1975, AFIPS Press, Montvale, NJ (1975).

14. Sturgul, J.R., and Merchant, M.J. Applied Fortran IV Programming, Wadsworth (1973).

15. Tulving, E. Subjective organization in free recall of "unrelated" words. Psychological Review 69 (1962), 344-354.

16. Weinberg, G.M. The Psychology of Computer Programming, Van Nostrand, New York (1971).

17. Wright, P., and Reid, F. Written information: some alternatives to prose for expressing the outcomes of complex contingencies. Journal of Applied Psychology 57, 2 (1973), 160-166.

Appendices

The experimental materials in these appendices are not the actual materials but are designed to give the reader a clearer impression of the nature of the experiments.

PILOT STUDY : MATERIALS

Read a sequence of data cards containing six digit numbers (punched in columns 1-6) into an array. There may be as many as 250 numbers. These cards are followed by a trailer card containing -99999 in columns 1-6. When the numbers have been read in and stored, read in the next data card which contains a target number. Search for the target number in the array and print out the number of the array position in which it is found with an appropriate message such as

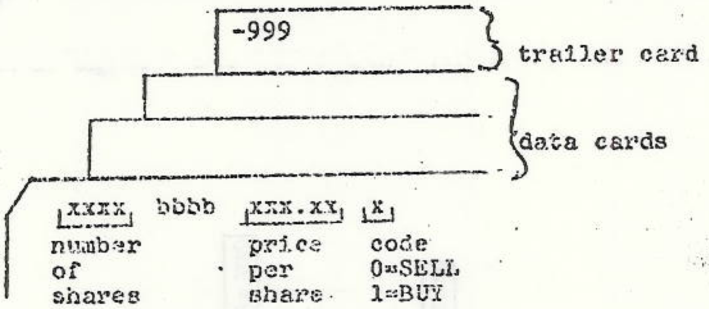
```
TARGET xxxxxx FOUND IN POSITION xxx
```

If the target is not found, print

```
xxxxxx NOT FOUND
```

EXPERIMENT I MATERIALS

Write a computer program to compute stock broker commissions. A series of data cards have been prepared in the following format:



The commission is computed as follows:

for sales: 2% if total value is less than $1000.00
1% if total value is greater than or equal to $1000.00

for purchases:

3% if total value is less than $500.00
2% if total value is greater than or equal to $500.00

Read each card, compute the commission and print the total value and the commission. When the trailer card is encountered print 'ALL DONE' and stop. Use FORMAT statement on input and output.

EXPERIMENT II : MATERIALS

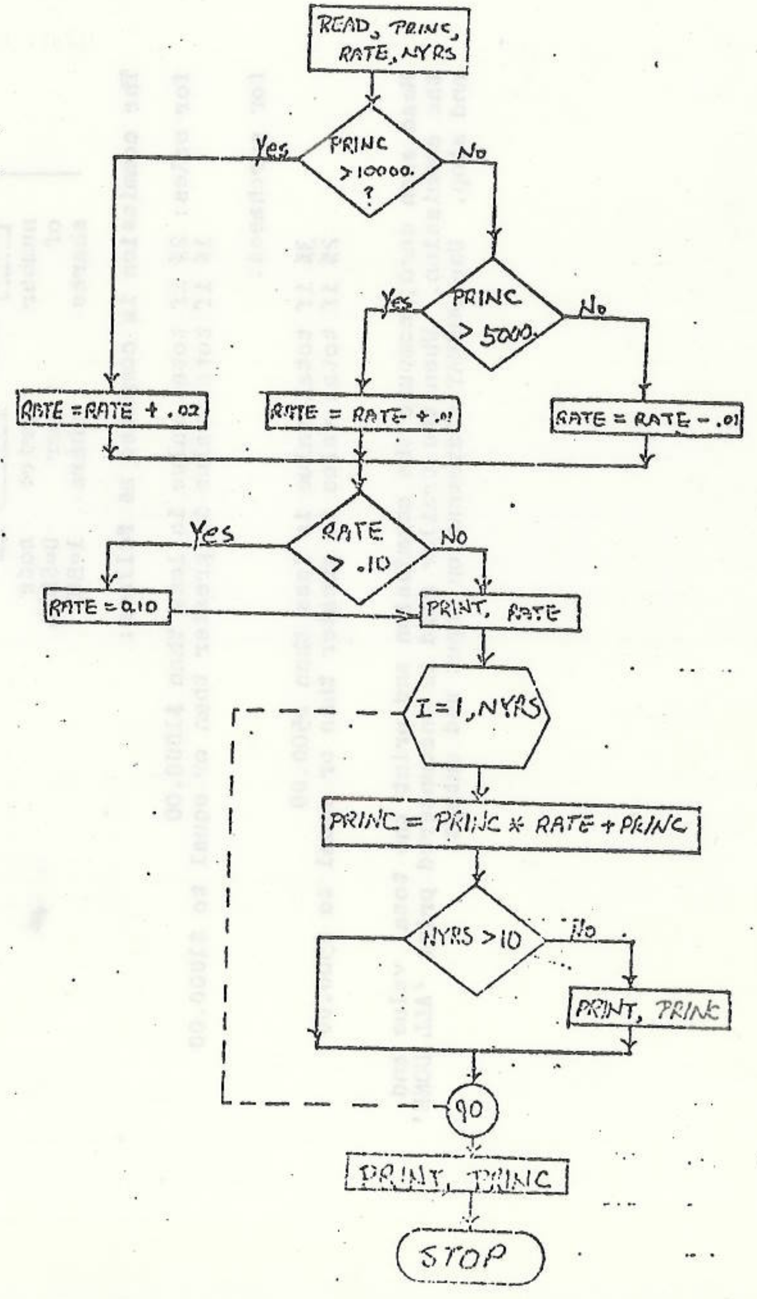
Part III - What is the output of statement 85 for the following sets of input: (2 points each)

```
15000.000.08 5      (1)____________________
b1000.000.06 7      (2)____________________
b7000.000.04 6      (3)____________________
b3000.000.252       (4)____________________
b5000.000.3812      (5)____________________
```

```
        READ(5,101)PRINC,RATE,NYRS
    101 FORMAT(F8.2,F4.2,I2)
        IF(PRINC.GT.10000.)GO TO 50
        IF(PRINC.GT.5000.)GO TO 20
        RATE=RATE-.01
        GO TO 80
     20 RATE=RATE+.01
        GO TO 80
     50 RATE=RATE+.02
     80 IF(RATE.GT..10)GO TO 70
     85 WRITE(6,102)RATE
    102 FORMAT(F5.2)
        DO 90 I=1,NYRS
        PRINC=PRINC*RATE+PRINC
        IF(NYRS.GT.10)GOTO 90
        WRITE(6,103)PRINC
    103 FORMAT(' PRINC ,F8.2)
     90 CONTINUE
        WRITE(6,104)PRINC
    104 FORMAT(' AMOUNT',F8.2)
        STOP
     70 RATE=.10
        GO TO 85
        END
```

Fill in the table with the number of times statement 20, 50, 70, and 90 get executed for each of the five input cards. (1 point each)

statement #	20	50	70	90
(1)				
(2)				
(3)				
(4)				
(5)				



Part IV – What is the output for the following sets of input data: (2 points each)

b3.510.25 (1) ____________________

b5.7b3.39 (2) ____________________

b5.0b5.06 (3) ____________________

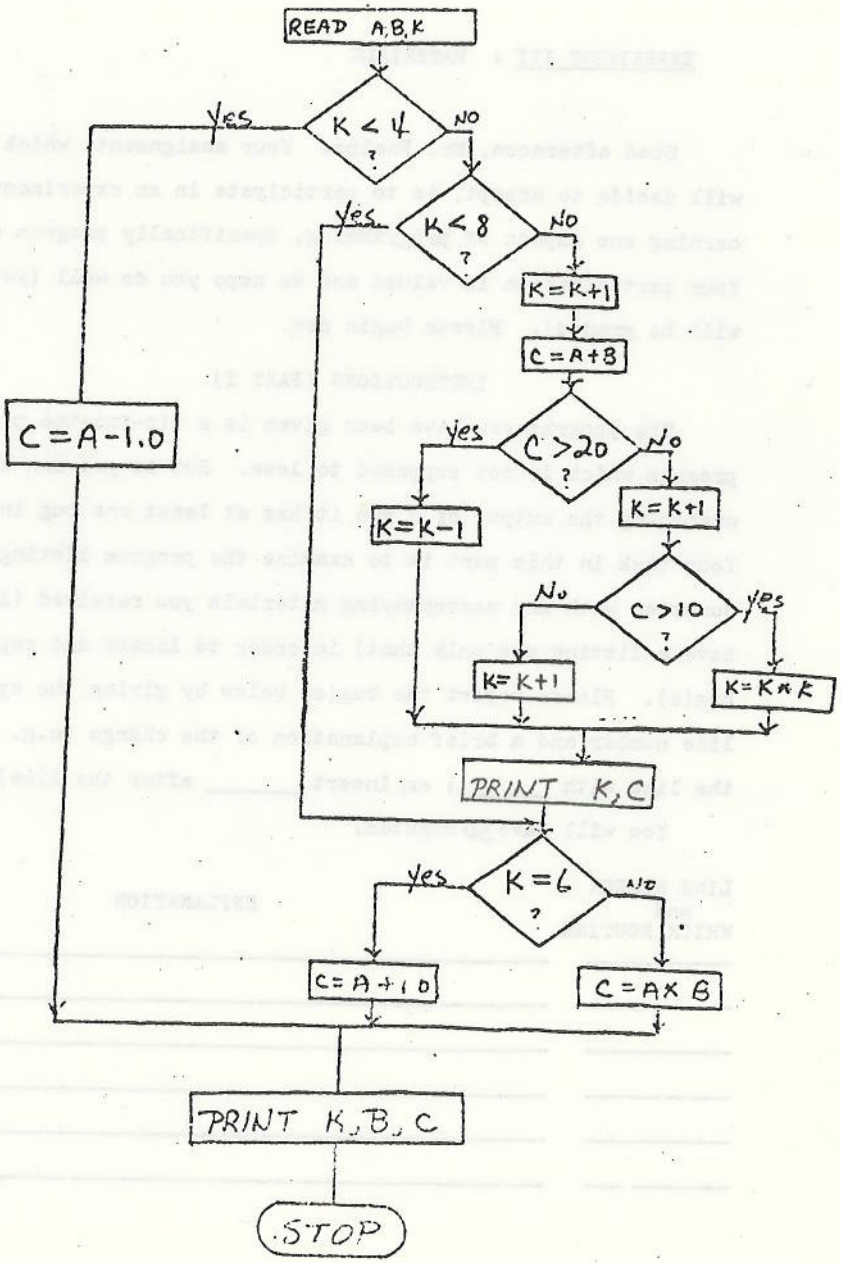
b7.217.09 (4) ____________________

22,4b1.72 (5) ____________________

```
    READ(5,102)A,B,K
102 FORMAT(F4.1,F4.1,I1)
    IF(K.LT.4)GO TO 90
    IF(K.LT.8)GO TO 50
    K=K+1
    C=A+B
    IF(C.GT.20.0)GO TO 60
    K=K+1
    IF(C.GT.10.0)GO TO 70
    K=K+1
10  WRITE(6,103)K,C
103 FORMAT(' RESULT',I3,F5.1)
    GO TO 50
70  K=K*K
    GO TO 10
60  K=K-1
    GO TO 10
50  IF(K.EQ.6)GO TO 55
    C=A*B
    GO TO 80
55  C=A+1.0
80  WRITE(6,104)K,B,C
104 FORMAT(' TEST',I3,F5.1,F5.1)
    STOP
90  C=A-1.0
    GO TO 80
    END
```

Fill in the table with the number of times statement 10, 50, 70, 80 get executed for each of the five input cards. (1 point each)

statement #	10	50	70	80
(1)				
(2)				
(3)				
(4)				
(5)				



EXPERIMENT III : MATERIALS

Good afternoon, Mr. Phelps. Your assignment, which you will decide to accept, is to participate in an experiment concerning one aspect of programming, specifically program debugging. Your participation is valued and we hope you do well (your answers will be graded). Please begin now.

INSTRUCTIONS (PART I)

The program you have been given is a tic-tac-toe playing program which is not supposed to lose. But as you can see by examining the output of a run it has at least one bug in it. Your task in this part is to examine the program listing in conjunction with any accompanying materials you received (if you only have a listing use only that) in order to locate and repair the bug(s). Please report the bug(s) below by giving the appropriate line number and a brief explanation of the change (e.g. replace the line with ______ ; or, insert ______ after the line).

You will have 50 minutes.

LINE NUMBER
and
WHICH ROUTINE   EXPLANATION

________ ________________________________

________ ________________________________

________ ________________________________

________ ________________________________

________ ________________________________

________ ________________________________

Name: Experiment Number III

INSTRUCTIONS (PART II)

Below are the bugs which are in the program. If they do not coincide with the bugs you found, please make the appropriate changes in your listing to fix the errors.

LINE NUMBER and WHICH ROUTINE	EXPLANATION
55 in TICTAC	should read: Z = Z .AND. M(I,J) .EQ. 0
62 in TICTAC	should read: Z = Z .OR. NCOL(J) .EQ. 1
21 in CELL	should read: ND = 2

After making any corrections necessary please answer the ten questions concerning your understanding of the program which begin on the next page. Answer each question by circling the correct response next to the question numbers supplied below. Again, use your listing and any accompanying materials in choosing your answers.

Once you have completed the questions please go on and fill out the questionaire which follows the questions. After you are finished please clip the four pages back together and hand them in. You may keep the listing and flowcharts.

You will have the rest of the period.

1.) A B C D E
2.) A B C D E
3.) A B C D E
4.) A B C D E
5.) A B C D E
6.) A B C D E
7.) (part i) A B C D E ; (part ii) A B C D E
8.) A B C D E
9.) A B C D E
10.) A B C D E

QUESTIONS

1.) What would happen if the second player entered 22 as his move?

A) an error
B) he would be told the move is not allowed
C) the program would continue normally
D) the program would stop
E) cannot tell without more information

2.) If the machine wins, how many times will subroutine CELL have been called?

A) NMOVE B) 8*NMOVE
C) 2*NMOVE-1 D) ≤16
E) cannot tell without more information

3.) How many cells are in the array M?

A) 3 B) 6 C) 9 D) 27 E) cannot tell without more information

4.) If the second player's first move was always to cell $M_{1,1}$ and his second move is selected randomly from the remaining available cells, what percent of the time will the machine win on it's third move?

A) 1/6 B) 1/2 C) 3/4 D) 5/6
E) cannot tell without more information

5.) If after the machine's first move, the second player took $M_{1,2}$, what would the machine's second move be?

A) $M_{1,1}$ B) $M_{2,1}$ C) $M_{3,1}$ D) any free corner
E) cannot tell without more information

6.) Using the same moves as in problem 5, if the second player's second move was to $M_{3,3}$, what would the machines third move be?

A) $M_{1,1}$ B) $M_{2,1}$ C) $M_{1,3}$ D) any free corner
E) cannot tell without more information

7.) If the board was as follows (machine = 0, second player = X)

```
 IXIO
-----
 IOI
-----
XIOIX
```

(i) What would the machine's move be?

A) $M_{1,1}$ B) $M_{2,1}$ C) $M_{2,3}$ D) the game is already over
E) cannot tell without more information

(ii) Using the same board as in part (i), what was the second player's first move?

A) $M_{1,2}$ B) $M_{3,1}$ C) $M_{3,3}$ D) cannot tell without more information

QUESTIONS (CONTINUED)

8.) If the second player's first move was to a corner, what is the greatest number of times the statement "DO 130 N = 1,8" (line 68) could be executed?

A) 0 B) 1 C) 2 D) 3
E) cannot tell without more information

9.) If on a certain move the machine "decides" to simply move in any vacant cell, how many times will subroutine CELL have been called before line 68 (DO 130 N = 1,8) is executed?

A) NMOVE B) 8*NMOVE
C) ≤16 D) 24
E) cannot tell without more information

10.) For this problem assume the second player forces the game into a draw and his first move was to $M_{1,3}$. How many moves to a corner will he have made in the game?

A) 1 B) 2 C) 3 D) 4
E) cannot tell without more information

QUESTIONAIRE

On the following questions please circle a number from 0 to 9 indicating your response to the question above each set of numbers and their labels coming as close to your true reaction as the rating scales permit.

1.) How well did you understand the program?

```
 0    1    2    3    4    5    6    7    8    9
not at                                        perfectly
 all
```

(Only answer the rest if you used a flowchart.)

2.) Which flowchart did you use? 1 page or 4 page ?

3.) How much did the flowchart help?

```
 0    1    2    3    4    5    6    7    8    9
not at                                        very much
 all
```

4.) How much did you use the flowchart?

```
 0    1    2    3    4    5    6    7    8    9
not at                                        very much
 all
```

Please go on to the next page.

QUESTIONAIRE (CONTINUED)

5.) Did the flowchart help in your understanding of the program?

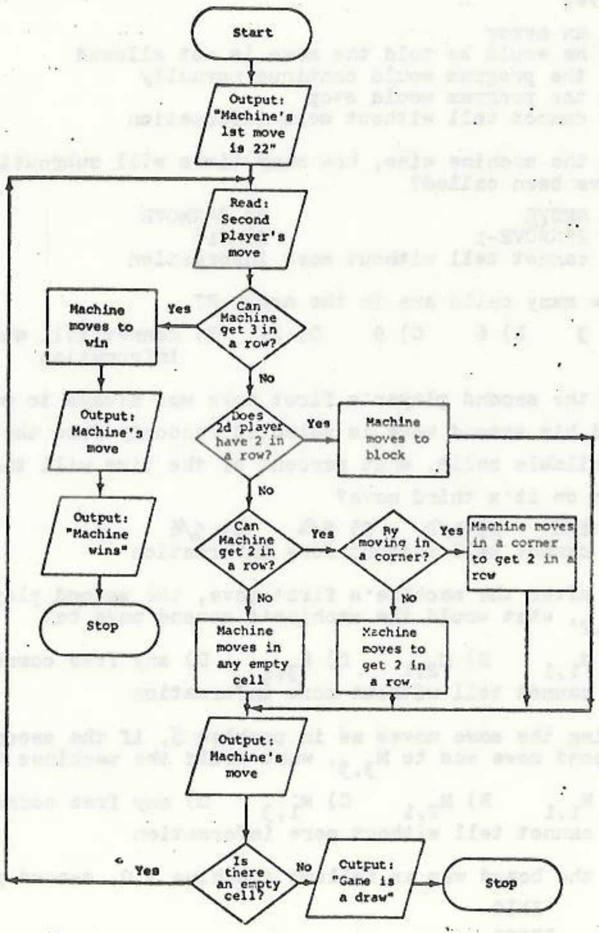
0 1 2 3 4 5 6 7 8 9
not at all … very much

6.) Did the flowchart help in finding the bugs?

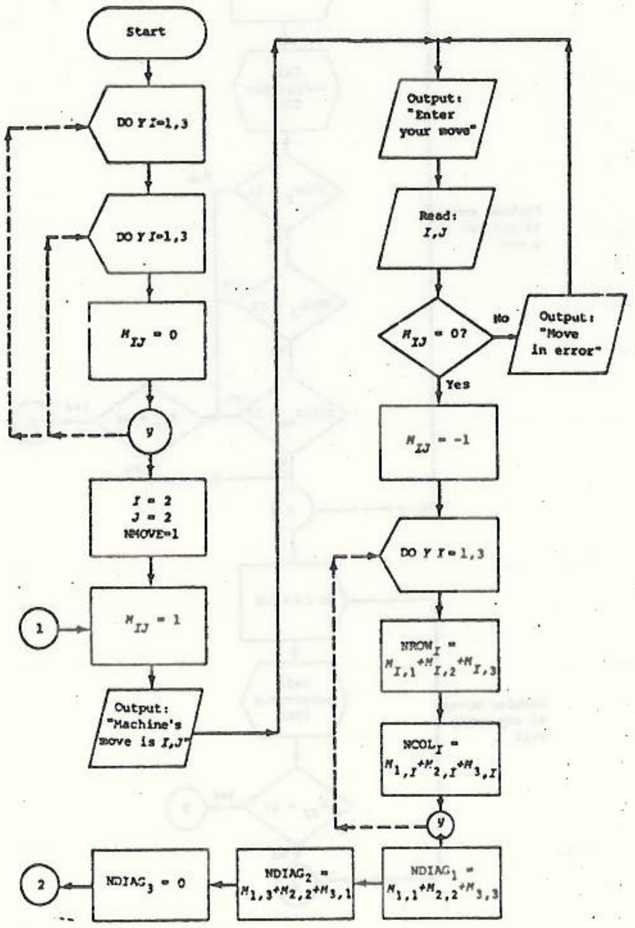
0 1 2 3 4 5 6 7 8 9
not at all … very much

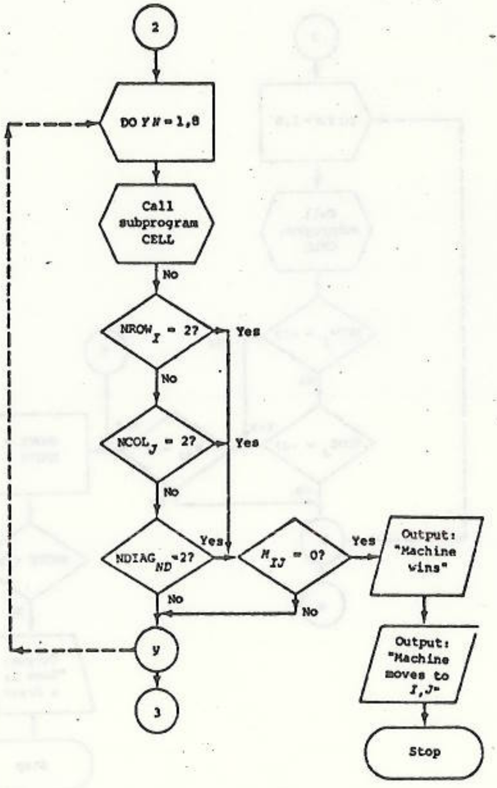
This is the end of the experiment. Thank you very much for your participation.

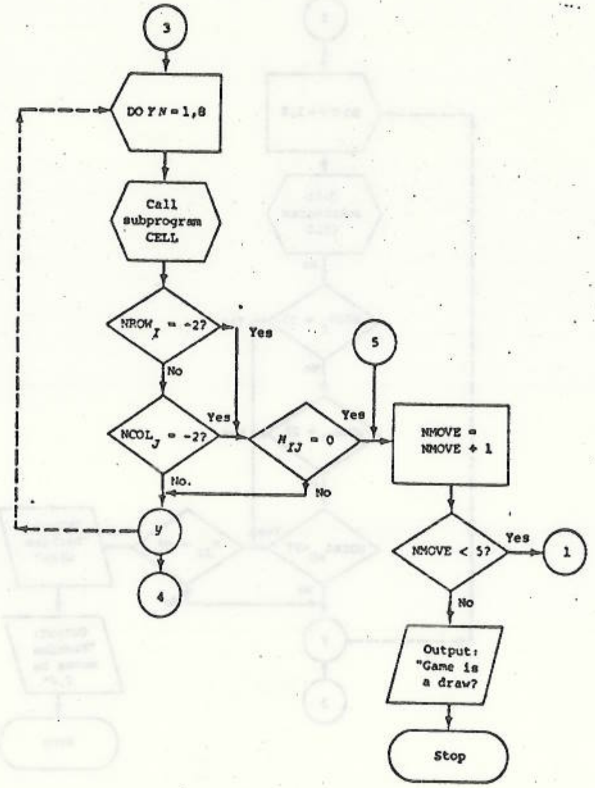
Macro Flowchart


Start
Output: "Machine's 1st move is 22"
Read: Second player's move
Can Machine get 3 in a row?
Yes
Machine moves to win
Output: Machine's move
Output: "Machine wins"
Stop
No
Does 2d player have 2 in a row?
Yes
Machine moves to block
No
Can Machine get 2 in a row?
Yes
By moving in a corner?
Yes
Machine moves in a corner to get 2 in a row
No
Machine moves to get 2 in a row
No
Machine moves in any empty cell
Output: Machine's move
Is there an empty cell?
Yes
No
Output: "Game is a draw"
Stop

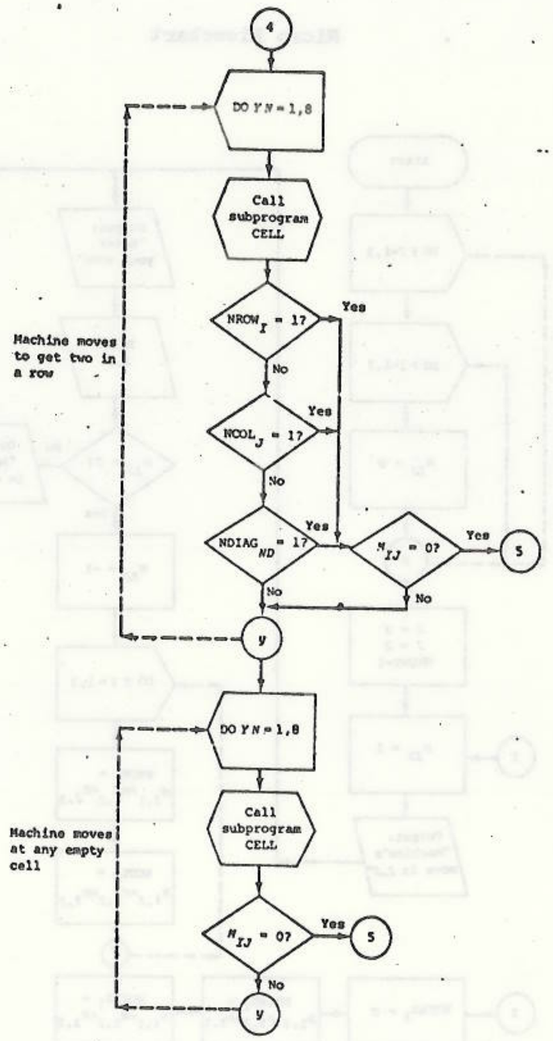
Micro Flowchart







3
DO Y N = 1,8
Call subprogram CELL
NROW$_I$ = -2?
Yes
No
NCOL$_J$ = -2?
Yes
No.
N_{IJ} = 0
Yes
No
5
y
4
NMOVE = NMOVE + 1
NMOVE < 5?
Yes
1
No
Output: "Game is a draw?
Stop



4
DO Y N = 1,8
Call subprogram CELL
NROW$_I$ = 1?
Yes
No
NCOL$_J$ = 1?
Yes
No
NDIAG$_{ND}$ = 1?
Yes
No
N_{IJ} = 0?
Yes
5
No
y
Machine moves to get two in a row
DO Y N = 1,8
Call subprogram CELL
N_{IJ} = 0?
Yes
5
No
y
Machine moves at any empty cell

```
         TICTAC       +-*/                            CDC 6600 FTN V3.0-V357

              PROGRAM TICTAC(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      C          PROGRAM TO PLAY TIC-TAC-TOE
      C          USES SUBROUTINES CELL AND BOARD
              DIMENSION M(3, 3), NROW(3), NCOL(3), NDIAG(3)
              LOGICAL Z
              COMMON N, I, J, ND
              DO 10 I = 1,3
              DO 10 J = 1,3
      10      M(I,J) = 0
              I = 2
              J = 2
              NMOVE = 1
      20      M(I,J) = 1
              WRITE(6,25) I, J
      25      FORMAT(1X, 'MACHINE MOVES AT', I2, ',', I2)
              CALL BOARD(M)
      30      WRITE(6,35)
      35      FORMAT(1X, 'ENTER YOUR MOVE')
              READ(5,40) I, J
      40      FORMAT(2I1)
              WRITE(6,41) I,J
      41      FORMAT(1X,2I1)
              IF(M(I,J) .EQ. 0) GO TO 50
              WRITE(6,45)
      45      FORMAT(1X, 'MOVE NOT ALLOWED')
              GO TO 30
      50      M(I,J) = -1
              DO 55 I = 1,3
              NROW(I) = M(I,1) + M(I,2) + M(I,3)
      55      NCOL(I) = M(1,I) + M(2,I) + M(3,I)
              NDIAG(1) = M(1,1) + M(2,2) + M(3,3)
              NDIAG(2) = M(3,1) + M(2,2) + M(1,3)
              NDIAG(3) = 0
              CALL BOARD(M)
      C          MACHINE MOVES TO GET THREE
      C          IN A ROW IF POSSIBLE
              DO 75 N = 1,8
              CALL CELL
              Z = NROW(I) .EQ. 2
              Z = Z .OR. NCOL(J) .EQ. 2
              Z = Z .OR. NDIAG(ND) .EQ. 2
              Z = Z .AND. M(I,J) .EQ. 0
              IF(Z) GO TO 100
      75      CONTINUE
              GO TO 105
      100     WRITE(6,25) I, J
              WRITE(6,101)
      101     FORMAT(1X, 'MACHINE WINS')
              STOP
      C          IF SECOND PLAYER HAS TWO IN A ROW, BLOCK
      105     DO 110 N = 1,8
              CALL CELL
              Z = NROW(I) .EQ. -2
              Z = Z .OR. NCOL(J) .EQ. -2
              Z = Z .OR. M(I,J) .EQ. 0
```

```
         TICTAC       R +-*/                            CDC 6600 FTN V3.0-V35

              IF(Z) GO TO 200
      110     CONTINUE
      C          MACHINE MOVES TO GET TWO IN A ROW IF POSSIBLE
              DO 120 N = 1,8
              CALL CELL
              Z = NROW(I) .EQ. 1
              Z = Z .OR. NCOL(I) .EQ. 1
              Z = Z .OR. NDIAG(ND) .EQ. 1
              Z = Z .AND. M(I,J) .EQ. 0
              IF(Z) GO TO 200
      120     CONTINUE
      C          MACHINE MOVES IN ANY VACANT CELL
              DO 130 N = 1,8
              CALL CELL
              Z = M(I,J) .EQ. 0
              IF(Z) GO TO 200
      130     CONTINUE
              STOP
      C          TEST TO SEE IF GAME IS A DRAW
      200     NMOVE = NMOVE + 1
              IF(NMOVE .LT. 5) GO TO 20
              WRITE(6,25) I, J
              WRITE(6,205)
      205     FORMAT(1X, 'GAME IS A DRAW')
              STOP
              END

      PROGRAM      TICTAC     R +-*/                      CDC 6600 FTN V3.0-V35

      SYMBOLIC REFERENCE MAP

    ENTRY POINTS
     4052   TICTAC

    VARIABLES          SN   TYPE                    RELOCATION
        1   I               INTEGER                      / /           2   J           INTEGER
     4337   M               INTEGER       ARRAY                        0   N           INTEGER
     4353   NCOL            INTEGER       ARRAY                        3   ND          INTEGER
     4356   NDIAG           INTEGER       ARRAY                     4336   NMOVE       INTEGER
     4350   NROW            INTEGER       ARRAY                     4335   Z           LOGICAL


    FILE NAMES           MODE
        0   INPUT                           2022   OUTPUT                         0   TAPE5        FM

    EXTERNALS                TYPE    ARGS
          BOARD                        1                            CELL                     0

    STATEMENT LABELS
        0   10                                      4065   20
     4101   30                                      4312   35       FMT
     4320   41       FMT                            4322   45       FMT
        0   55                                         0   75
     4326   101      FMT                            4211   105
        0   120                                        0   130
     4331   205      FMT
    SUBROUTINE  CELL        R +-*/                        CDC 6600 FTN V3.0-V359

                  SUBROUTINE CELL
                  COMMON N, I, J, ND
                  IF(N .NE. 1) GO TO 2
                  I = 1
                  J = 1
                  ND = 1
                  RETURN
          2       IF(N .NE. 2) GO TO 3
                  I = 1
                  J = 3
                  ND = 2
                  RETURN
          3       IF(N .NE. 3) GO TO 4
                  I = 3
                  J = 3
                  ND = 1
                  RETURN
          4       IF(N .NE. 4) GO TO 5
                  I = 3
                  J = 1
                  ND = 1
                  RETURN
          5       IF(N .NE. 5) GO TO 6
                  I = 1
                  J = 2
                  ND = 3
                  RETURN
          6       IF(N .NE. 6) GO TO 7
                  I = 2
                  J = 3
                  ND = 3
                  RETURN
          7       IF(N .NE. 7) GO TO 8
                  I = 3
                  J = 2
                  ND = 3
                  RETURN
          8       IF(N .NE. 9) STOP
                  I = 2
                  J = 1
                  ND = 3
                  RETURN
                  END

    SUBROUTINE  CELL        R +-*/                        CDC 6600 FTN V3.0-V359

    SYMBOLIC REFERENCE MAP

ENTRY POINTS
    1  CELL

VARIABLES       SN  TYPE               RELOCATION
    1  I            INTEGER               / /          2  J            INTEGER
    0  N            INTEGER               / /          3  ND           INTEGER

STATEMENT LABELS
   10  2                                   17  3
   33  5                                   42  6
   56  8
```

```
    SUBROUTINE  BOARD       R +-*/                        CDC 6600 FTN V3.0-V

                  SUBROUTINE BOARD(MOVES)
                  DIMENSION MOVES(3,3),PBOARD(3,3)
                  DATA BLANK/1H /, X/1HX/, O/1HO/
                  DO 10 I = 1,3
                  DO 10 J = 1,3
                  IF(MOVES(I,J))4,6,9
          4       PBOARD(I,J) = X
                  GO TO 10
          6       PBOARD(I,J) = BLANK
                  GO TO 10
          9       PBOARD(I,J) = O
                  GO TO 10
         10       CONTINUE
                  DO 20 I = 1,2
                  WRITE(6,15) (PBOARD(I,J),J = 1,3)
         15       FORMAT(1X,A1,'I',A1,'I',A1)
                  WRITE(6,16)
         16       FORMAT(1X,5('-'))
         20       CONTINUE
                  WRITE(6,25) (PBOARD(3,J),J = 1,3)
         25       FORMAT(1X,A1,'I',A1,'I',A1,/)
                  RETURN
                  END

    SUBROUTINE  BOARD       R +-*/                        CDC 6600 FTN V3.0-V

    SYMBOLIC REFERENCE MAP

ENTRY POINTS
    2  BOARD

VARIABLES       SN  TYPE               RELOCATION
   71  BLANK        REAL                                104  I            INTEGER
  105  J            INTEGER                               0  MOVES        INTEGER
   73  O            REAL                                106  PBOARD       REAL
   72  X            REAL

FILE NAMES          MODE
       TAPE6         FMT

STATEMENT LABELS
    0  4            INACTIVE              24  6
   27  10                                 74  15         FMT
    0  20                                101  25         FMT

STATISTICS
  PROGRAM LENGTH      122B      82
```

```
MACHINE MOVES AT 2, 2
 I I
-----
 IOI
-----
 I I

ENTER YOUR MOVE
31
 I I
-----
 IOI
-----
XI I

MACHINE MOVES AT 1, 1
OI I
-----
 IOI
-----
XI I

ENTER YOUR MOVE
33
OI I
-----
 IOI
-----
XI IX

MACHINE MOVES AT 1, 3
OI IO
-----
 IOI
-----
XI IX

ENTER YOUR MOVE


32
OI IO
-----
 IOI
-----
XIXIX

MACHINE MOVES AT 1, 2
MACHINE WINS
```

EXPERIMENT IV : MATERIALS

```
1          INSTRUCTIONS
    THE EXPERIMENT IN WHICH YOU ARE PARTICIPATING IS
 CONCERNED WITH TESTING HUMAN FACTORS IN COMPUTER PROGRAMMING.
     THE PRESENT EXPERIMENT IS TESTING YOUR ABILITY TO
 MODIFY EXISTING PROGRAMS.  IN THIS PACKET YOU WILL FIND A
 PROGRAM LISTING, REFERENCE MAP, SAMPLE OUTPUT AND YOUR FIRST
 MODIFICATION ASSIGNMENT.  WHEN WRITING YOUR MODIFICATIONS
 FEEL FREE TO WRITE SUBROUTINES OR FUNCTIONS IF YOU WISH,
 BUT PLEASE INDICATE WHERE IN THE PROGRAM YOUR MODIFICATIONS ARE
 TO BE INSERTED.
 YOU NEED NOT RECOPY THE ENTIRE PROGRAM.  YOUR  MODIFICATIONS WILL
 BE JUDGED ON THE EFFICIENCY OF CODE, CLARITY, BUT MOST
 HEAVILY ON RUNABILITY AND CORRECTNESS.  YOUR PACKET CONTAINS THREE
 MODIFICATIONS.   EACH MODIFICATION DESCRIPTION ASKS YOU TO NOTE THE TIME
 YOU STARTED THAT MODIFICATION AND WHEN YOU FINISHED.   PLEASE ACCURATELY
 RECORD THE TIME.  YOU WILL NOT BE GRADED ON HOW LONG YOU REQUIRE FOR A
 PARTICULAR MODIFICATION.  WHEN FINISHED WITH ONE, PROCEED TO THE NEXT.
 ****  ASK ANY QUESTIONS NOW   ******
 DO NOT BEGIN UNTIL INSTRUCTED TO DO SO, THANKYOU
```

```
  PROGRAM     GRADER    R +-*/                              CDC 6600 FTN V3.0-V359 OPT=1

                    PROGRAM GRADER(INPUT,OUTPUT)
                    DIMENSION NAME(3), IADRS(3), NMECSE(3)
              C
              C
              C   GRADER GENERATES SEMESTER GRADE REPORTS FOR IUCS UNIV..
              C   THE PROGRAM ACCEPTS AS INPUT TWO MASTER CARDS AND ANY
              C   NUMBER OF DATA CARDS.  THE MASTER CARDS ARE DEFINED AS
              C       ID - A STUDENTS SOCIAL SECURITY NUMBER
              C       NAME - 30 CHARACTER NAME
              C       IADRS - A 30 CHARACTER ADDRESSS
              C       IZIP - ZIP CODE
              C       CRHRS - TOTAL CREDIT HOURS TO DATE
              C       CRPTS - TOTAL CREDIT POINTS TO DATE.
              C  IUCS UNIV. GRADES ON THE TRADITIONAL 4.0 SCHEME IE. 4.0 IS AN 'A'.
              C
              C  THE INDIVIDUAL CLASS REPORTS ARE DEFINED TO BE
              C       NMECSE - THE COURSE NAME
              C       NMEDPT - THE NAME OF THE DEPARTMENT THE COURSE IS OFFERED BY.
              C       NUMCSE - THE COURSE NUMBER DESRIBED IN THE CATALOG OF CLASSES
              C       NUMSEC / THE SECTION NUMBER ASSIGNED BY THE REGISTRAR
              C       CRDHR - THE NUMBER IF CREDIT HOURS THE CLASS IS
              C       LGRD - THE LETTER GRADE RECEIVED FOR THE CLASS
              C
              C
              C  READ MASTER CARD
              100   READ 1,ID,NAME,IADRS,IZIP
              1     FORMAT(I9,3A10,3A10,I5)
                    IF(ID .EQ. (-1)) GO TO 300
                    READ 2,CRHRS,CRPTS
              2     FORMAT(F5.1,F5.1)
                    PRINT 101
              101   FORMAT('1',84(1H*))
                    PRINT 115
                    PRINT 103,ID , NAME, IADRS,IZIP
              103   FORMAT(' *',1X,I9,3X,3A10,3X,3A10,1X,I5,1H*)
                    PRINT 115
                    PRINT 104
              104   FORMAT(' *',' COURSE NAME',19X,'DEPT NUMBER SECTION HOURS',
                   1 'GRADE-REMARKS',12X,1H*)
                    PRINT 115
              C   INITIALIZE SEMESTER COUNTERS
                    SEMHR = 0.0
                    SEMPTS = 0.0
              C    DETERMINE THE NUMBER OF HOURS AND POINTS ACCUMULATED FOR THE SEM
              C   START-READ LOOP FOR INDIVIDUAL CASS REPORTS
              50    READ 3,IFLAG,NMECSE,NMEDPT,NUMCSE,CRDHR,LGRD,NUMSEC
              3     FORMAT(I1,3A10,A2,I3,F2.0,A2,I6)
                    IF(IFLAG)10,10,20
              C   PROCESS CARD
              C    INDIVIDUAL COURSE REPORT
              10    PRINT 114,NMECSE,NMEDPT,NUMCSE,NUMSEC,CRDHR,LGRD
              114   FORMAT(' *',3A10,3X,A2,1X,I6,1X,I7,F5.1,5X,A1,21X,1H*)
                    IF(LGRD .EQ. 1HA) SEMPTS = SEMPTS + 4.0 * CRDHR
                    IF(LGRD .EQ. 1HB) SEMPTS = SEMPTS + 3.0 * CRDHR
                    IF(LGRD .EQ. 1HC) SEMPTS = SEMPTS + 2.0 * CRDHR
 1  PROGRAM     GRADER    R +-*/                              CDC 6600 FTN V3.0-V359 OPT=1

                    IF(LGRD .EQ. 1HD) SEMPTS = SEMPTS + 1.0 * CRDHR
                    IF((LGRD .EQ.1HP) .OR. (LGRD .EQ. 1HW)) GO TO 40
                    SEMHR = SEMHR + CRDHR
              40    GO TO 50
              20    DO 150 I=1,10
              150   PRINT 115
              115   FORMAT(' *',82X,'*')
              C    PRINT THE SEMESTER STATISTICS AND UPDATED OVERALLL STATS
                    PRINT 116
              116   FORMAT(' *',3X,' GPA',3X,'CREDIT POINTS',3X,'CREDIT HOURS',3X,
                   1   'SEM. GPA',3X,'SEM. POINTS',3X,'SEM. HOURS',6X,'*')
              C  COMPUTE SEMESTER AVG AND NEW GPA
                    SEMGPA = SEMPTS / SEMHR
                    CRHRS = CRHRS + SEMHR
                    CRPTS = CRPTS + SEMPTS
                    GPA = CRPTS / CRHRS
                    PRINT 117,GPA,CRPTS,CRHRS,SEMGPA,SEMPTS,SEMHR
              117   FORMAT(' *',F6.3,5X,F6.0,11X,F6.0,5X,F8.3,6X,F6.0,7X,F6.0,10X,'
                    PRINT 111
              111   FORMAT(' ',84(1H*))
                    GO TO 100
              300   STOP
```

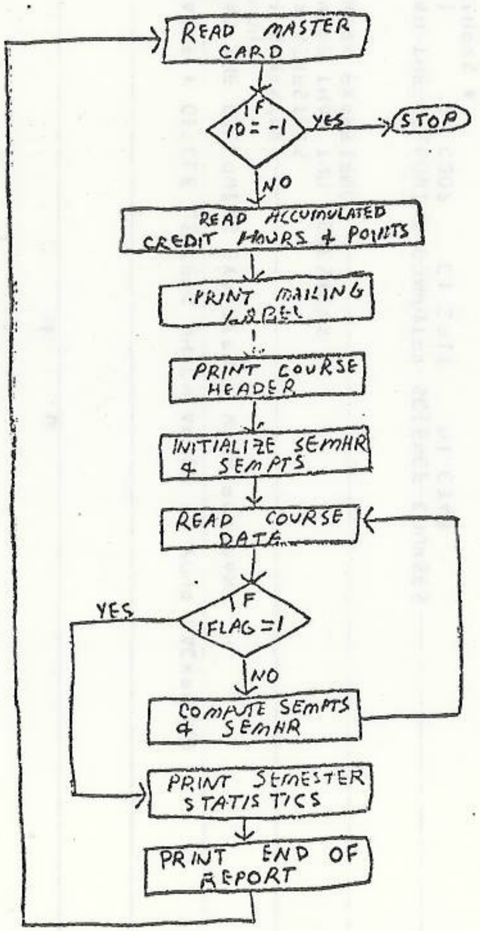
```
1            MODIFICATION 1

     THE REREGISTRAR WISHES TO INCLUDE MORE ADMINISTRATIVE INFORMATION
   ON THE THE IUCS  GRADE REPORTS.  TWO GRADES HAVE BEEN ADDED TO THE EXISTING
   GRADES.  A NEW GRADE IS 'N' WHICH INDICATES AN INSTRUCTOR HAS NEGLECTED TO
   REPORT A GRADE.  IF SUCH A GRADE IS ENCOUNTERED, THE COURSE SHOULD
   BE LISTED IN THE GRADE REPORT BUT NOT COUNTED TOWARDS THE GPA OR
   SEMESTER GPA.  ALSO, A REMARK SHOULD BE INCLUDED INDICATING THE
   STUDENT SHOULD SEE THE INSTRUCTOR.  THE SECOND ADDITION IS 'I' WHICH
   INDICATES THE INSTRUCTOR BELIEVES THE STUDENT HAS NOT COMPLETED THE
   REQUIREMENTS FOR THE COURSE.  A GRADE OF 'I' SHOULD BE TREATED AS 'N'
   EXCEPT THE REMARK SHOULD REMIND THE STUDENT TO MAKE UP THE WORK.
```

```
1            MODIFICATION 2

     THE DEAN OF CSCI HAS CONVINCED THE REGISTRAR TO INCLUDE NOTICES
   FROM HONOR SOCITIES.  THE FOLLOWING HONOR SOCIETIES AND THEIR
   REQUIREMENTS ARE LISTED BELOW.
        SEP -  ANYONE COMPLETING THEIR FRESHMAN YEAR (IE. MORE THAN 30 HOURS
         BUT LESS THAN 50 HOURS) IS ELIGIBLE FOR INDUCTION IF THEIR
         OVERALL GPA IS ABOVE 3.0

        IUCS  -  THE IUCS STIPEND AWARD FOR SCHOLARSHIP IS PRESENTED
         TO ANY STUDENT WHO HAS A 3.3 OVERALL GPA OR BETTER.  THE
         RECOGNITION CARRIES A $100 HONORARY SCHOLARSHIP PER
         SEMESTER.

        DEAN  -  THIS AWARD IS PRESENTED TO THOSE STUDENTS WHO ATTAINED
         A SEMESTER GPA OF 3.45 OR BETTER.

        THE ABOVE SOCIETIES AND HONARIES HAVE BEEN GRANTED GRADE
   REPORT STATUS.  ANY STUDENT WHO MEETS ANY OR ALL OF THE ABOVE
   CRITEREON FOR THE SOCIETIES SHOULD BE NOTIFIED ON THEIR GRADE
   REPORT WITH AN APPROPRIATE MESSAGE BRIEFLY DESCRIBING THE AWARD
   OR TELLING HIM WHO TO CONTACT.
```

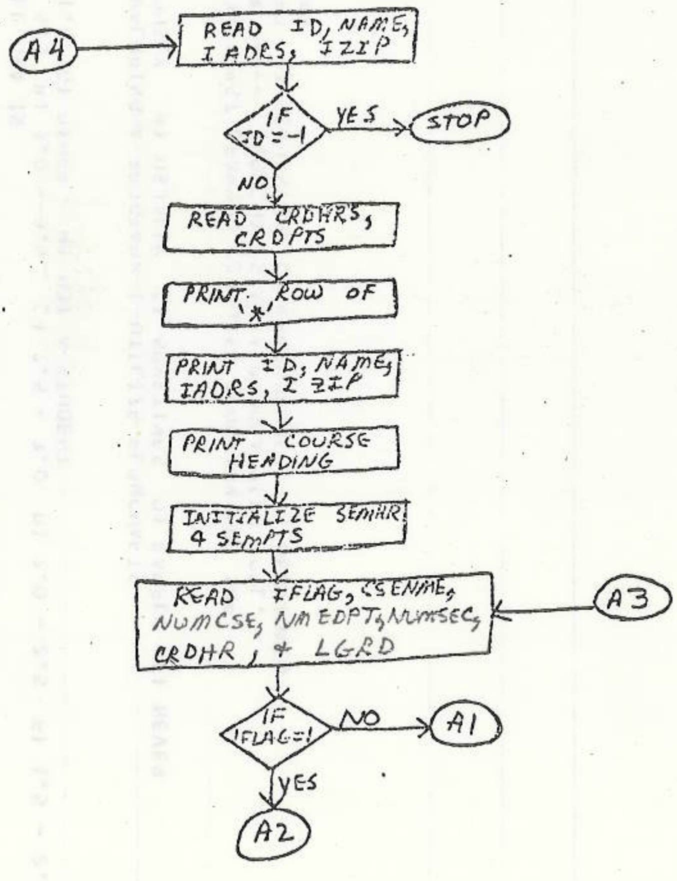
```
1            MODIFICATION 3
     THE REGISTRAR HAS GONE NUTS.  HE HAS ALLOWED PHI BERA KAPPA
   GRADE REPORT STATUS.  PHI BERA KAPPA ACCEPTS ANY SENIOR (IE. 90 - 132 CREDIT
   HOURS) WHICH IS IN THE TOP 10 PER CENT OF THE SENIOR CLASS.  THE REGISTRAR
   ESTIMATES THAT THE AVERAGE NUMBER OF GRADUATING SENIORS IS ABOUT 100.
    IF A SENIOR HAS MET PHI BERA KAPPA'S GRADE REQUIREMENTS, THEY SHOULD BE
   NOTIFIED ON THEIR GRADE REPORT.
```

```
1   QUESTIONNAIRE * CIRCLE LETTERS WHICH APPLY TO YOUR BACKROUND

  1* I HAVE HAD THE FOLLOWING EXPERIENCE WITH FORTRAN
          A) NONE
          B) 1 SEMESTER
          C) 2 SEMESTERS
          D) MORE THAN TWO SEMESTERS
          E) WORK EXPERIENCE

  2* I HAVE HAD THE FOLLOWING COMPUTER SCIENCE COURSES
       A) C201      B) C202     C) C311     D) C343
       LIST OTHERS *

  3* I HAVE WRITTEN A PROGRAM CLOSELY RESEMBLING THIS PROGRAM
          A) YES    B) NO

  4) I CONSIDER MYSELF A(N) --------- FORTRAN PROGRAMMER.
      A) EXCELLENT   B) GOOD   C)FAIR   D) POOR

  5) I AM A(N)
          A) UNDERGRADUATE     B) GRADUATE

  6* I AM A COMPUTER SCIENCE MAJOR (GRAD / UNDERGRAD )
          A) YES    B) NO

  7* MY COMPUTER SCIENCE GPA IS *
     A) 3.5 - 4.0  B) 3.0 - 3.5   C) 2.5 - 3.0  D) 2.0 - 2.5  E) 1.5 - 2.0
    F) 1.0 - 1.5  G) OTHER   H) NOT A STUDENT

  8* MY OVERALL GPA IS
     A) 3.5 - 4.0  B) 3.0 - 3.5   C) 2.5 - 3.0  D) 2.0 - 2.5  E) 1.5 - 2.0
    F) 1.0 - 1.5  G) OTHER   H) NOT A STUDENT

  9)  WHEN DEVELOPING A PROGRAM, I UTILIZE FLOWCHARTS
    A) RELIGIOUSLY   B) USUALLY  C) SOMETIMES  D) RARELY  E) NEVER

  10)  WHEN YOU FIRST LEARNED COMPUTER PROGRAMMING, THE
    INSTRUCTOR -----  FLOWCHARTS AS A PROGRAMMING TOOL.
    A) REQUIRED  B) REQUESTED  C) RECOMMENDED  D) MENTIONED
     E) IGNORED
```

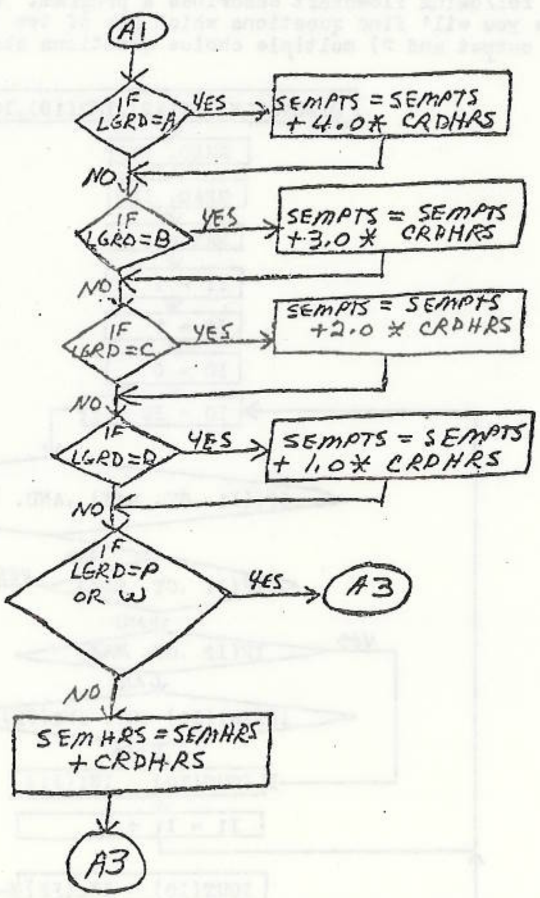
MACRO - FLOWCHART



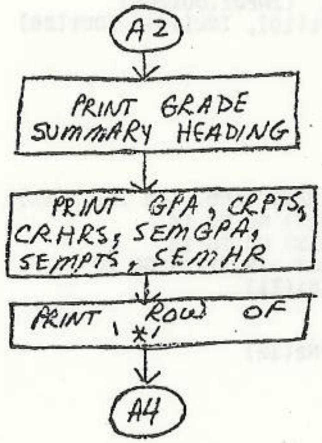
MICRO - FLOWCHART


A4
READ ID, NAME, IADRS, IZIP
IF ID=-1
YES
STOP
NO
READ CRDHRS, CRDPTS
PRINT ROW OF '*'
PRINT ID, NAME, IADRS, IZIP
PRINT COURSE HEADING
INITIALIZE SEMHR & SEMPTS
READ IFLAG, CSENME, NUMCSE, NMEDPT, NUMSEC, CRDHR, & LGRD
A3
IF IFLAG=1
NO
A1
YES
A2

MICRO- FLOWCHART


A1
IF LGRD=A
YES
SEMPTS = SEMPTS + 4.0 * CRDHRS
NO
IF LGRD=B
YES
SEMPTS = SEMPTS +3.0 * CRDHRS
NO
IF LGRD=C
YES
SEMPTS = SEMPTS +2.0 * CRDHRS
NO
IF LGRD=D
YES
SEMPTS = SEMPTS + 1.0 * CRDHRS
NO
IF LGRD=P OR W
YES
A3
NO
SEMHRS = SEMHRS + CRDHRS
A3

MICRO- FLOW CHART

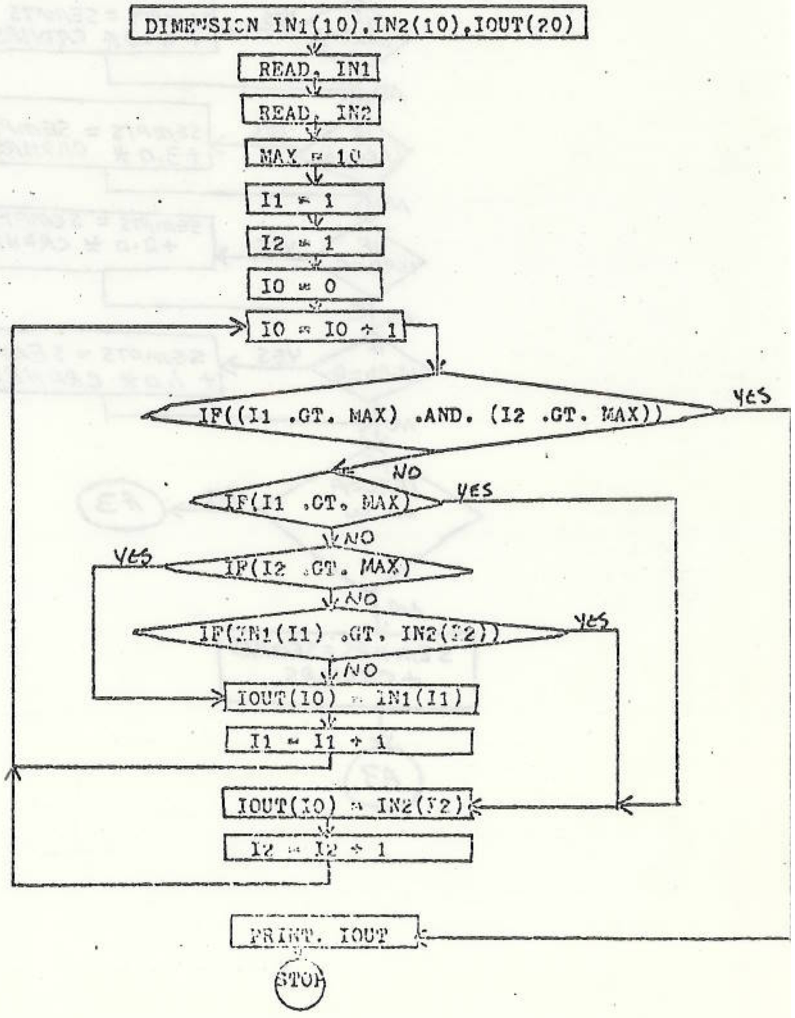

A2
PRINT GRADE SUMMARY HEADING
PRINT GPA, CRPTS, CRHRS, SEMGPA, SEMPTS, SEMHR
PRINT ROW OF '*'
A4

EXPERIMENT V : MATERIALS

The following is a program. On the following page you will find questions which are of two types: 1) supply the output and 2) multiple choice questions about the program.

```
        PROGRAM QIII  (INPUT,OUTPUT)
        DIMENSION IN1(10), IN2(10), IOUT(20)
        READ, IN1
        READ, IN2
        MAX = 10
        I1 = 1
        I2 = 1
        IO = 0
20      IO = IO + 1
        IF((I1 .GT. MAX) .AND. (I2 .GT. MAX)) GO TO 50
        IF(I1 .GT. MAX) GO TO 10
        IF(I2 .GT. MAX) GO TO 15
        IF(IN1(I1) .GT. IN2(I2)) GO TO 10
15      IOUT(IO) = IN1(I1)
        I1 = I1 + 1
        GO TO 20
10      IOUT(IO) = IN2(I2)
        I2 = I2 + 1
        GO TO 20
50      PRINT, IOUT
        STOP
        END
```

The following flowchart describes a program. On the following page you will find questions which are of two types: 1) supply the output and 2) multiple choice questions about the program.



1) If the 10 values read into IN1 were:
1 3 5 7 9 11 13 15 17 19, respectively;
and the 10 values read into IN2 were:
2 4 6 8 10 12 14 16 18 20, respectively;
what would be the 20 values output, place your values in the appropriate box.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20

2) If all the values of IN1 are less than the first value of IN2,

a) all the values of IN2 will appear in the output before the values in IN1.
b) all the values of IN1 will not appear in the output.
c) all the values of IN2 will not appear in the output.
d) all the values in IN1 will appear in the output before the values in IN2.

3) If the input arrays, IN1 and IN2, are sorted in decreasing order, the output array, IOUT, will be in

a) decreasing order
b) no apparent order
c) increasing order

4) If the 10 values read into IN1 were:
1 2 8 20 23 24 25 26 27 28, respectively;
and the 10 values read into IN2 were:
3 4 5 6 7 30 35 40 45 50, respectively;
what would be the 20 values output, place your values in the appropriate box.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20

5) To have the program produce output in decreasing order,

a) the input must be in decreasing order.
b) the input must be in increasing order.
c) the input must be in increasing order and the program must be changed, *
d) the input must be in decreasing order and the program must be changed.*

* If the program must be changed, make your modification on the previous page

Analysis of Variance Table for Experiment II

Analysis of Variance for Percent Correct

FACTOR	df	MEAN SQUARE	F	P
Problem (Problem 1 vs 2)	1	30.25	5.8	<.05
Group (Flowchart on 1 vs 2)	1	30.25	1.8	
Within Subjects Error	48	16.15		
Problem X Group	1	2.89		
Between Subjects Error	48	5.17		

Analysis of Variance Tables For Experiment IV

Analysis of Variance for Time Measure

FACTOR	df	MEAN SQUARE	F	P
University	1	33.075		
Type of FC	2	8.0083		
Modification	1	.2083		
Univ X Type of FC	2	29.5750		
Univ X Modification	1	10.2083		
Type of FC X Modification	2	20.3583		
Within Subjects				
Error	54	17.5713		
Univ X Type of FC X Modification	2	1.2583		
Between Subjects				
Error	54	21.8491		

Analysis of Variance for Percent Correct

FACTOR	df	MEAN SQUARE	F	p
Univ	1	32.033	9.9	< .01
Type of FC	2	1.525		
Modification	1	12.033	5.78	< .05
Univ X Type of FC	2	1.0083		
Univ X Modification	1	10.800	5.22	< .05
Type of FC X Modification	2	2.6083		
Within Subjects				
Error	54	3.3278		
Univ X Type of FC X Modification	2	5.9250	2.84	< .10
Between Subjects				
Error	54	2.0759		

Analysis of Variance Table for Experiment V

Analysis of Variance for Percent Correct

FACTOR	df	MEAN SQUARE	F	p
Type of Quiz	2	913.8		
Type of Question	1	0.24		
Within Subject				
Error	48	2582.5		
Type of Quiz X Type of Question	2	321.4		
Between Subject				
Error	48	612.2		