INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Grossman, Elaine Sue

A COMPARISON OF INSTRUCTIONAL METHODS EMPLOYING GROUP AND INDIVIDUAL PROGRAMMING IN AN INTRODUCTORY COMPUTER SCIENCE COURSE

Columbia University Teachers College ED.D. 1983

A COMPARISON OF INSTRUCTIONAL METHODS EMPLOYING GROUP AND INDIVIDUAL PROGRAMMING IN AN INTRODUCTORY COMPUTER SCIENCE COURSE

by

Elaine Sue Grossman

Dissertation Committee:

Professor Bruce Vogeli, Sponsor
Professor J. Philip Smith

Approved by the Committee on the Degree of Doctor of Education

Date MAY 9 1983

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Education in Teachers College, Columbia University

1983

ABSTRACT

A COMPARISON OF INSTRUCTIONAL METHODS EMPLOYING GROUP AND INDIVIDUAL PROGRAMMING IN AN INTRODUCTORY COMPUTER SCIENCE COURSE

Elaine Sue Grossman

The purpose of this study was to examine student programming proficiency, perseverance, efficiency, and attitude in an introductory computer science programming course under three different methods of instruction.

The investigation took place at a suburban, private, four-year college that services approximately 10,000 students under a policy of open enrollment. Three classes of an undergraduate introductory course in structured FORTRAN were selected. One class (N = 21) received instruction in the traditional mode employing individual programming; the second class (N = 20) received instruction in the experimental mode employing fixed heterogeneous student programming groups; and the third class (N = 19) received instruction in the experimental mode employing variant heterogeneous student programming groups.

Analysis of the data revealed the following results:

1. The experimental classes achieved higher average programming assignment grades than the control class, although the differences on the midterm and final examinations, taken individually, on which the experimental classes scored lower than or similarly to the control class, were not found to be significant.

2. Students in the experimental classes completed significantly more programming assignments on time than did students in the control class.

3. Students in the experimental classes spent more time writing progams and using terminals than students in the control class. The number of runs required per assignment for most students in the experimental classes was less than that required by students in the control class. However, none of the efficiency measures produced significant results.

4. In a posttreatment survey, students in the experimental classes demonstrated significantly more positive attitudes toward computers and computer programming than did students in the control class.

No significant differential effects were produced by any of the methods on measures of programming proficiency or programming efficiency. However, in introductory courses, in which computer literacy is a main course objective, the more positive effects substantiated for measures of programming perseverance and attitude in the classes employing student programming groups lead to the

conclusion that these instructional methods are worthy of consideration for courses designed for student populations as described in the study.

ACKNOWLEDGMENTS

I wish to express my appreciation to my sponsor, Dr. Bruce Vogeli, for his guidance and assistance. His suggestions and perceptive insights helped me at every stage of this research.

I am also grateful to the members of my Doctoral Committee, Dr. J. Philip Smith, Dr. Jane Monroe and Dr. Robert Taylor, for their valuable comments and criticisms.

Finally, I wish to express my love and appreciation to my husband, Harvey, and my children, Jennifer and Benjamin, for their sacrifices and support in helping me accomplish my educational goals.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Need

The area of computer science education, relatively new to the school community, has now become the focus of increased attention and research (Basili & Reiter, 1981; Lemos, 1977). Lucas (1976) claims that as computer literacy becomes necessary for the survival of the individual in modern society, all students will need to deal effectively with complex computer applications and have the ability to balance the opportunities and dangers presented in the use of computers.

Responding to this, undergraduate and graduate programs across the nation are requiring their students to be familiar with, if not proficient in, computer programming. Virtually all undergraduates at Dartmouth, all undergraduates at Harvard (Fiske, 1980), and majors in the fine arts, social sciences, physical sciences, decision sciences, natural sciences, English, engineering, mathematics, and business at many academic institutions are being required to come in contact with some aspect of computer science. The specific requirements vary but the overall picture is clear: To become a well-equipped graduate for today's society, one

must be computer literate (Johnson, Anderson, Hansen, & Klassen, 1980; Thomas & Carroll, 1979). Consequently computer faculties and facilities within the academic institutions are facing an increase in enrollment of computer science majors and a need to offer service-oriented computer science courses for other units within the college.

Coupled with this, the decreasing college population has forced many schools to open and expand programs and schedules to attract and accommodate the non-traditional as well as the traditional student. As a result, computer science courses that in the 1960's were available to and accepted by mathematics and science majors only, are now being offered to the entire spectrum of the college community. Thus, academic computer facilities and faculties are being required to respond to a growing audience of undergraduates, representing a wide range of interests and levels of academic maturity and ability, all while operating under the financial and physical constraints of their institutions.

Especially sensitive to this predicament are the introductory level programming courses that are service courses as well as filtering agents for beginning computer science majors. Cognitively, these courses are designed to develop the student's skill in writing computer-oriented solutions to varying types of problems; and, affectively, they are directed to foster a

"constructive and positive attitude toward the potential of computers" (Sackman, 1970, p. 16). All undergraduate introductory courses claim similar proficiency and attitudinal objectives; however, courses in computer programming additionally demand inordinate amounts of departmental resources (Lemos, 1977) and student time (University of Nebraska Newletter, Note 1).

Porter and Nesa (1975), Khailany (1977b), and Lemos (1979b), among many other computer instructors, find that beginning programming students spend time waiting for computer terminals to be free. Often the programs the students write don't work and are full of errors (bugs), requiring more time to search for errors (debug) and correct errors (edit) (Barnett, 1978; Conway, 1974; Eckberg, 1976; Rosenberg, 1976; Sebaugh, 1976). Thus, normal demands of any introductory course are further strained by the investment the programming student, especially the non-traditional student, who may have job and/or family responsibilities, must make in terms of the limited resource--time (Deimel & Pozefsky, 1979; Schribner, 1974).

Brooks (1975) states that the writing and running and rewriting and rerunning of a program become a process that causes tension, anxiety and frustration. "Cheating" often becomes the fast, necessary solution, with students copying someone else's program that did work or having someone else write theirs. This, together with the guilt

of the act and non-learning of the situation, is a poor resolution at best (Alford, Hsia & Petry, 1977; Conway, 1974; Grier, 1981; Mathis, 1974; Miller, 1981; Sackman, 1970).

These circumstances have ramifications for the instructors of introductory computer programming courses. They find themselves having to explain repeatedly simple syntactical rules and point out common errors in logic to their students and, in general, seeing frustration and anxiety discourage more of their students than they can encourage (Ahmed & Bardos, 1976; Weinberg, 1971).

Considering the budgetary constraints, the faculty and facility limitations and the student problems accompanying the rapid expansion of undergraduate computer programming courses, investigation of instructional methods that are effective and efficient in producing students skilled in programming and with positive attitudes toward computers is a primary concern for research (Ahlgren, Sapega & Warner, 1978; Cheng, 1976; Gillett, 1976; Martin & Docouis, 1964).

Purpose

The purpose of this study is to investigate the effects of the instructional method employing student programming groups on student achievement and attitude in an introductory level computer programming course.

The group organization used in advanced programming

computer courses claims to expose its members to the mechanisms of a real-world situation by working toward a common goal, adhering to deadlines and encountering personality conflicts (Dinerstein, 1975; Freeman, 1976; Mackey & Fosdick, 1979; Mize, 1976; Perry & Weymouth, 1975; Plum & Weinberg, 1974; Ruschitzka, 1977; Tam & Busenberg, 1977). Educationally, however, professional training, although important, cannot be accepted as sufficient justification for use of this method in lower level courses. In such courses there is a diversity of students that does not exist in advanced courses. Some students may be taking the course as an introduction to a major, while others may be fulfilling a requirement for another major or taking the course for self-enlightenment. In these lower level computer courses the use of group programming requires substantiation by cognitive gains and/or promotion of positive attitudes toward computers and computer programming for all these students.

Major impetus to explore the learning of computer programming using group activities has come from Weinberg's (1971) advocacy of "egoless programming" within programming assemblages. "Egoless programming" occurs when the process of producing a program in an individual, secretive mode is discouraged and the process of producing a program in an open, shared manner is encouraged (Lemos, 1979b). Supported and aided by the

group in designing, coding and debugging, the individual writes his/her program.

Weinberg, although mainly concerned with the professional programmer in industry and not with the amateur programmer in schools, states that the main difference between the two is in the user of the final product (i.e., the program) and that many of the advantages of the process of programming within a group are applicable to any learning situation.

The aim of the student programming group would be: (a) to establish formally a routine of working together in the analysis and design of solutions to programming assignments and (b) to create and maintain a willingness on the part of each member of the group to peruse a group member's coded solution in exchange for receiving the same from that member. Students programming in the group environment would still write programs individually; but now they would have other programmers to whom they could turn for help and who, at the same time, might be turning to them.

To substaniate these claims this study investigates the programming achievements and attitudes of the student in the normal flow of informal student exchanges as they compare to the achievements and attitudes of the student operating within the formal structure of the group, where such exchanges are not only encouraged but are demanded. Two experimental instructional methods in computer

science education, combining lecture with group programming, are explored. One method employs fixed heterogeneous student programming groups and the other employs variant heterogeneous student programming groups.

The experiment is divided into three comparisons. Comparisons 1 and 2 contrast the programming skills and attitudes of students receiving instruction in the traditional mode, lecture and individual programming, to those of students in each of the experimental instructional sections, in which lecture is combined with group programming. Comparison 3 contrasts the programming skills and attitudes of students receiving instruction in the two experimental methods: programming within fixed heterogeneous student groups and programming within changing heterogeneous student groups.

Specific questions to be investigated are:

1. Do students who receive instruction in the fixed group or variant group or traditional setting achieve a higher level of programming proficiency?

2. Does the individual operating within the fixed programming group or the variant programming group or programming alone exhibit greater programming perseverance?

3. Do students who work within the fixed group structure or the variant group structure or alone make more efficient use of their time (pertinent to the course) and computer facilities?

4. Do students who program within the fixed group or variant group or traditional environment exhibit a more positive attitude toward computers and computer programming?

Three sections of an undergraduate course in introductory computer programming form the sample for the experiment. All three sections are taught by the investigator and cover the same material, from the same text and lecture notes and at the same pace. The students in one section are taught using the traditional method of lecture and individual programming; the students in the second section receive a similar lecture but are directed to program within fixed heterogeneous groups; and the students in the third section receive a similar lecture but are directed to program within heterogeneous groups that are changed for every programming assignment.

The experiment begins with the third programming assignment and includes the fourth, fifth and sixth programming assignments and the individually taken midterm and final examinations.

A survey entitled Attitude Toward Computers and Computer Programming (see Appendix F) is taken by all students at the beginning of the semester. At the end of the experiment students retake the survey and are encouraged to enumerate their general reactions to the course.

To aid in the interpretation of the data obtained in Comparisons 1, 2 and 3, participation in a seventh programming assignment, a team project, is required of all the students in the study.

CHAPTER II

REVIEW OF LITERATURE RELATED TO GROUP PROCESS AND ATTITUDE

Group Process

Definition

The casual assemblage of individuals to perform a task does not necessarily define a functioning group in a socio-psychological sense. Consider for example, the failure of baseball players to act cohesively as a team in the All-Star game (Weinberg, 1971) or the inability of the members of the United Nations to act as a unified force. It is the extent of cohesiveness in a group that has major implications for communication and influences decision-making processes and other task performances that characterize a well-functioning group.

A cohesive group can be defined as an efficient and effective group that is recognized by its members as providing the means to accomplish goals more easily and quickly than they could alone. In order for the group to function successfully, each member of the group must be an active participant and group members must work for a common goal because they feel some direct personal gain or satisfaction derived from the completion of the task (Aleck, 1959; Bruce, 1959; Gates, 1948; Jewell & Reitz, 1981; Johnson, Johnson, Johnson & Anderson, 1976; Piaget,

1971; Thiabut & Kelley, 1959).

However, if a group has one or more members who have personal objectives which take precedence over group objectives, the group tends to become non-functioning for all members. Members of such collectives guard their assets jealously and are not aware of the liabilities of themselves to others. The group lacks the necessary cohesion and the members become apathetic or indifferent.

> This failure of one or more members to share the group goals affects the group performance; not only through that member's share but through a reduced performance on the part of others, who invariably perceive the division within the group or the indifference of one of the members. (Weinberg, 1971, p. 72)

Kelley and Thiabut (1954) discuss an experiment conducted by Gurnee in 1937 in which group process was compared to individual process. Some individuals worked together as groups for trials #2-#6 and made significantly fewer errors than did the individuals working alone. However on the seventh trial the group members worked as individuals and made almost as many errors as those who had been working individually throughout the experiment. The conclusion was that:

> Although groups performing as groups were more accurate than individuals, group members showed no evidence of superior learning in their subsequent individual performances. (p. 122)

Gurnee attributed the correct decision promoted by the group over the individuals as resulting from: (a) the scattering of individual uncorrelated errors

cancelling each other out by the correct majority opinion of the group and (b) the individuals in the group being subjected to social influences (pressures) causing a modification of their own solutions during the course of group discussion. The rejection of incorrect ideas that escape the notice of the individual when working alone and the changing of opinion as a result of social interaction in the direction of the consensus of the group worked to produce the correct group solution.

In the analysis of individual decisions made within a group as compared to individual decision-making, the personalities of the group members are considered to play a leading role in predisposing members to self-confidence. It has been determined that self-confidence combined with the self-weighing process, in which the individual's contributions or suggestions are withheld according to the degree of certainty that the individual feels about them, greatly influence the success of the individual within the group problem-solving situation.

Kelley and Thiabut state that with certain types of tasks, among which they list the more complex intellectual responses of problem-solving (e.g., concept formation, detecting relationships, analytic reasoning, etc.), the presence of others makes subjects more cautious and constrained and responses are produced with greater delay and are more commonplace in nature.

This claim of the deleterious effect of social factors is supported by the studies of Zajonc (1962) in which "together" situations are reported to increase the liklihood of the occurrences of responses most rehearsed and decrease the liklihood of less practical ones. There is a rapid homogenization of viewpoint, usually led by a knowledgeable member, overwhelming potentially valuable contributions from minorities (Maier & Solem, 1952) and prehaps causing loss of motivation and alienation of members.

Negative consequences of social interaction in group problem-solving, also cited by Lamm and Trommsdorff (1973), for example, presence of others not conducive to verbalization of novel or creative ideas, fear of criticism, distraction by ideas of others, presence of dominant member, are not supported in a study done by Philipsen, Mulac and Dietrich (1979).

The latter researchers examined the effect of social interaction in small group problem-solving on idea generation by a group member. The investigation, using four-member groups, found no difference in a measure of post-idea generation discussion after vocalization tasks written by a real group (i.e., fixed formal group) and a nominal group (i.e., joining of four individuals' results).

Philipsen, Mulac and Dietrich agree that the individual working alone is not exposed to social

distractions, but report that the benefits derived from cognitive stimulation by other group members far outweigh the diversions that possibly exist in a group.

Use In Education

Employing group process in the classroom is not a new or original idea. The educational psychologist, Bell (1978b) states that:

> Although expository teaching and teacher-centered demonstrations are good strategies for introducing facts, skills and concepts, they are not particularly effective in promoting in-depth learning of . . . principles . . . . Students can learn a great deal . . . from each other and can identify their own areas of confusion by attempting to explain skills concepts and principles to other students. (p. 205)

Bruner (1966) offers further support for the utilization of student groups in education and states that:

> Mental development depends upon systematic and structured interactions between the learner and the teachers; a student's "teachers" are other students, parents, school-teachers, or anyone who chooses to instruct the learner. (p. 6)

Piaget (1973) criticizes the

> traditional school that recognizes only the social exchange that is connected with a teacher, who is a kind of absolute ruler in control of moral and intellectual truth over each individual student. In such a school collaboration among the students and even direct communication among them are . . . excluded from classwork and homework because of the examination atmosphere and grades to be met. (p. 108)

Continuing, Piaget states that since collective living has been shown to be essential to the full

development of the personality in all its facets--even the more intellectual, an "active" school environment should be established providing opportunities for the students to alternate betwen individual work and working in groups.

According to both Bruner and Piaget, the one thing that many instructors tend not to do is to exploit the unique abilities which students have for teaching each other. On many occasions, students are better able to learn concepts by discussing them with each other and explaining them to each other than through exclusive instruction from the teacher (Bell, 1978a; McKuen & Davidson, 1975).

Agreeing with Piaget and Bruner, Brown (1975) defines education to be essentially a social process that occurs continually as a result of interaction with other human beings. Although a student may learn by doing something alone, the meaning of what is being done is realized only through the student's interaction with other people (McKuen & Davidson, 1975; Stanford & Roark, 1974).

Bruner labels this interaction between the student and others as "reciprocity" and links it to the student's will to learn.

> Finally, a word about one last intrinsic motive that bears closely upon the will to learn. Perhaps it should be called reciprocity, for it involves a deep human need to respond to others and to operate jointly with them toward an objective . . . . It is

> about as primitive an aspect of human behavior as we know . . . . Its exercise seems to be its sole response through reciprocity to other members of one's species. Where joint action is needed, where reciprocity is required for the group to attain an objective, then there seem to be processes that carry the individual along into learning, sweep him into a competence that is required in the setting of the group. We know precious little about this primitive motive to reciprocate, but what we do know is that it can furnish a driving force to learn as well. (p. 125)

Small Group Problem-Solving In The Classroom

In support of small group learning in the classroom, Johnson and Johnson (1979) conducted an experiment in an elementary science class. Forced by a lack of supplies, the use of group interaction in science classrooms has long been the accepted pedagogical approach. Although the experimenters state the intention to form heterogeneous groups, in which students would have different backgrounds, perspectives and skills for "the most powerful problem-solving situation" (p. 27), random methods were used to form four-member groups.

The results of the study indicate that cooperative learning caused students to share information, generate alternative ideas, invent tests to try out each other's ideas and sharpen their inference through discussion. Besides finding an increase in cognitive learning, the benefits reported by group interaction were student self-esteem and a more positive attitude toward science.

One specific advantage for the instructor (concurring with findings by Davidson, 1974) was the time

that was saved from having to repeat directions, as someone in each group will have listened enough to know what to do.

Davidson, Agreen and Davis (1978) employed the small group discovery method in a junior high mathematics class. Hoping that incompatiable personality combinations would be avoided (see studies done by Hoffman, 1979), students were permitted to form their own groups. This method resulted in forming a few homogeneous groups and, although they may allow the slower or faster learner to move along at his or her own pace, "groups consisting solely of very slow learners are a disaster more often than not" (p. 25).

Davidson, Agreen and Davis found that the classroom atmosphere became relaxed and informal when help was readily available. Questions were freely asked and answered, and even the shy student found it easy to be involved. The teacher-student relationship tended to be more relaxed and pleasant than in a traditional approach. In addition students were able to maintain a high level of interest in the small group activities. Some slow learners liked math more, or at least hated it less, than in teacher-directed approaches.

The disadvantages reported basically concerned the difficulty involved in forming effective working groups, covering the material (contradicting findings of Goldberg, Note 2) and persistant absenteeism. Those

students who frequently missed work were not supported by the other group members who were reluctant to take the time to help the absentee. This contradicts the results of Davidson (1974) who states that when using the group situation to learn mathematics, the group was an invaluable aid in helping returning students "catch up" on the work they had missed while absent.

Seeking a change from passive learning situations accompanied by too much time spent answering trivial questions and having extroverted students dominate class discussions and overwhelm introverted students who had unasked questions, Artzt (1979) formed teams in a secondary level mathematics class.

Artzt used this method in five, 50 minute classes a day for two years. Students divided themselves into teams of four, five or six members that would change at the completion of a given unit of work. The class would have five to seven teams and each handed in one copy of the previous night's homework, corrected by the group and designated as the team's homework, besides submitting individual homework assignments for comment, correction and grades.

The instructor attempted to foster group interaction by penalizing teams for incorrect statements in team homework, missing homework from team members, late arriving team members, receiving the lowest team average on a quiz and any disciplinary action necessitated by a

team member.

Artzt found the first grouping to be of friends but after the first assignment, weaker groups demanded equalization.

> The desire to create teams of equal strength forced teams to find members of different abilities; such balancing was to everybody's advantage. (p. 507)

Since the class consisted of the better students (i.e., those students who were on the Regents track), the differences between the students was not wide, and so it was not considered a burden for them to help each other. During the eighteenth week of the year the teams had stabilized to everyone's satisfaction.

It was determined that the students were getting better grades in mathematics and the classes were scoring higher on the Regents exam than other classes and than the instructor's own classes in previous years.

Another result reported by the experimenter was an improved attitude toward math in slower students. How the attitude improvement was assessed is not explained nor does Artzt comment on the increase in paper work for the instructor or on the reaction of students to becoming morally responsible for their team members' actions.

Also experimenting with students in secondary education, Garibaldi (1979) conducted a small-scale investigation to assess the affective benefits of using cooperative and group goal structures on two problem-solving tasks. Working with 92 high school

students randomly assigned to one of four experimental conditions (two-person pure cooperation, two-person group with intergroup competition, individualization, two-person interpersonal competition), Garibaldi found that students who worked in group-cooperation and intergroup competition performed best, expressed greater certainty about their answers and demonstrated more enjoyment of tasks than students who worked alone (interpersonal competition or individualization).

Employing a Likert-type questionnaire, Garibaldi's results support those found by Johnson and Johnson (1979) and Goldberg (Note 2) that students who cooperate perform better, have a more favorable attitude toward peers and tasks and indicate a higher degree of commitment to their answers than do students who compete or work individually.

In addition, Garibaldi concluded that individual competition appeared to be "ineffective in performance improvement for low achievers" (p. 794); while low ability students who worked with high ability partners improved in performance (number of correct answers) significantly.

Similar results are to be found in the studies done by Goldman (1965) in which undergraduate students of different abilities worked in paired groups on problems in the Wonderlic Intelligence Test. The partner paired with the student above his level was reported to have

improved significantly over the individual who was paired with a partner at or below his level.

Garibaldi does not consider the problem of the more knowledgeable group member dominating the group answer, thereby necessitating the learning of the low-ability student to be assessed by other means.

Research using problem-solving cooperative groups for mathematics instruction on the undergraduate level was done by Goldberg (Note 2). Heterogeneous teams of four or five abstract algebra students were formed. These groups worked on assignments together in class and outside. They could divide each assignment into sections and have each member of the group do one section or they could elect to work on the entire solution together. Either way they were required to discuss, edit and submit a single written solution to the instructor.

Goldberg found that:

> The grades for the semester determined in this fashion were higher than those usually given to students taking the same class with me. The group assignment scores were generally better than those of individuals, but so were the midterm and final scores and these were obtained individually. Rather than wasting lecture time for group discussions (as most instructors fear will happen in innovative teaching), the subject matter covered in the course was the most extensive in my experience. (p. 5)

A departmental evaluation questionnaire elicited positive reactions from students toward the group study method. Students felt they were better able to "digest" the material in the pleasant socially-oriented

environment but were burdened by the time requirement for group meetings outside of the classroom.

Cooperative problem-solving is viewed by Lightner (1981) as the exposing of the individuals in the group milieu to aspects of a problem situation which may or may not have had the same weight or imprint to them in an isolated and individualized milieu.

A clarification process occurs for the group participant when he/she is forced, by interaction, to recite the problem-solving steps which will lead to a plausible solution and when the alternative solutions of group members point out the relevances and irrelevances of the problem.

> This distillation process, while possibly leading to the same solution as some participants might conceive out of isolation or individual study, does have a distinct learning advantage of systematizing the problem-solving approach. It gives each group member a more intuitive understanding of the issues and forces which bear upon the solution of the problem. (p. 5.)

Lightner claims that this understanding can be translated into a more rapid achievement of knowledge and subject matter.

To substaniate these benefits of group dynamics, Lightner conducted a comparative experiment in intermediate undergraduate accounting courses using traditional lecture format in Fall 1977 and group problem-solving in the Spring 1978. The experimenter expected that the group milieu would change students'

passive learning attitudes and foster sharing and out-of-classroom discussions that would improve the students' comprehension of the subject matter while reflecting the professional real world role expected of the accounting major.

The same four examinations were administered to both the Fall and Spring classes, but to encourage group interaction, Lightner offered the students in the experimental classes an additional maximum of five points from the examination grades of their group members. These additional points for group effort were obtained by finding 5% of the average of the team examination scores. In this manner it was possible for a team member to achieve a score of 105 on the examinations.

Testing for homogenity in the group and non-grouped classes, Lightner found no signifigant differences in the averages of their previous accounting grades, yet her results showed a marked difference in learning (i.e., a higher mean examination score between the two).

Lightner agrees with Garibaldi that the "beneficiaries of the group process learning appear to be the academically poorer students" (p. 8). Examining the range of final exam scores which were two standard deviations from the mean, demonstrated that the performances of the weaker students in the groups markedly increased over those in the non-grouped situation. This contradicts the results of a study

conducted by McLeod and Adams (1979), in which slow students had trouble keeping up with the pace set by their group and there were students who fell behind too far to profit from group discussion.

The better students in Lightner's study were motivated to help the weaker students because points on part of their exam grades depended on their performances, yet results found that better students performed well under either group or non-group settings.

Lightner lists the additional time commitments required as group members and the existence of divergent study habits as possible problem areas when employing group process in the classroom, but states that the improved academic performance of poorer students, and the creation of more cooperation and interest needed for professional accounting activities form a strong case for the implementation of participatory group dynamics in accounting education.

Similar results are to be found in a recent study by King (1978) in nursing education. King's course, incorporating Gestalt therapy, confluent education and group process was based on Brown's (1975) statement that although a student may learn by doing something alone, the meaning of the task is realized only through interaction with other people.

To humanize the approach of nursing students to diabetic cases, the students were randomly divided into

five-person groups to discuss patient diet and medication, personal reactions and, in general, to give support to each other. Student responses initially were mixed, but as the term progressed the students became more involved with each other's work. Ideas were exchanged and feelings were shared. At the end of the course the responses were positive.

Although no differences were noted in student performances, King reports that the

> substanial interaction in the task groups . . . provided a much finer supply of information than that which emerged from the larger group, where only two or three students could present their information and the rest passively listen. (p. 24)

Attitudes

Attitudes and Behavior

Attitude, being a hypothetical construct, defies a single, final definition (Scott, 1968; Shaw & Wright, 1967). Allport (1935) defines attitude as a mental and neural state of readiness, organized through experience and exerting a directive or dynamic influence upon the individual's responses to all objects and situations to which it is related.

Disagreeing with Allport, Bain (1928) and Horowitz and Horowitz (1938) view attitudes as essentially the response rather than a set to respond. Studies by Sherif and Sherif (1968) find attitudes to be collections of

responses and define attitude, operationally, as the

> individual's set of categories for evaluating a stimulus domain in interaction with other persons and which relate the individual to various subsets within the domain with varying degrees of positive or negative effect. (p. 115)

Attitude, defined in the above manner, can then be inferred from the consistent characteristic behavior of the individual. However, attitudes may also be viewed in causal relationships. Znaniecki (1939) attempts to make a distinction between attitudes, opinions and knowledge and define attitude to be a process of individual consciousness which determines real or possible activities of the individual in the social world. Further distinctions are made by Hartley (1968), who finds it difficult to disengage attitude from other processes that serve similar functions and states that the study of attitudes must "involve the understanding of emotions and motivations" (p. 92).

The study of attitudes occupies a central place in all social-psychological research. No theory of social behavior can be complete without incorporation of attitude functioning and it is doubtful that "complex social behavior can be predicted without knowledge of attitudes" (Shaw & Wright, 1967, p. 14). Whether the definition of attitude is focused on the potential to respond or on the response itself, attitudes are linked to the behavior of the individual.

In the studies done by Fishein and Ajzen (1974,

1975), attitudes and subjective norms assume a central position in the etiology of behavior by means of their influences on intentions, and not by their direct impact on behavior.

The subjective norm, which is a measure of the influence of the social environment on an individual's behavior, corresponds to the individual's beliefs regarding whether those referents, who are important to him or to her, think that he or she should perform a given behavior.

Bentler and Speckart (1979) expand on this theory and offer a generalized attitude behavior-relations model (see Figure 1) in which attitude is proposed to have a direct influence on behavior in addition to an "indirect influence on behavior by means of cognition-conation intentions" (p. 455).

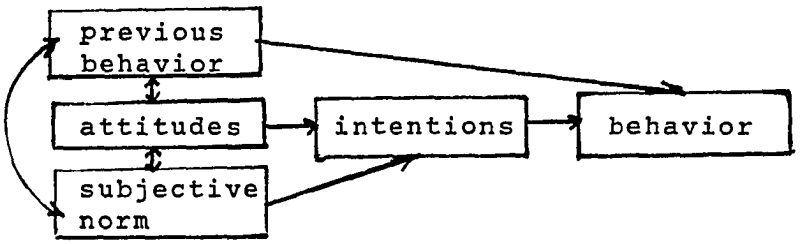


Figure 1. A generalized attitude behavior-relation model.

Zimbardo and Ebbesen (1970) view attitudes as enduring predispositons, ones which are learned rather than innate. Thus even though attitudes are not momentarily transcient, they are susceptible to change. Consequently, directing a change of the underlying attitude should produce more enduring changes in behavior than would be produced by trying to change directly only the behavior in question.

Attitude Change and Groups

Zimbardo and Ebbesen (1970) state that since attitudes are susceptible to change, all the techniques relevant to learning any material should be relevant to learning and changing attitudes. Basically, methods of attitude change rely upon the assumption that change comes out of conflict, discrepancy, inconsistency or discontent with the status quo.

Usually when the researcher (instructor) wishes to change an attitude in a favorable direction, an attempt is made to bring about acceptance of the proposition that the attitude object possesses positively valued attributes (Hovland, Janis, & Kelley, 1953; Kiesler, Collins & Miller, 1969). Within the group milieu this occurs when the group becomes a source of need-satisfacion for the individual student. By identifying with the group, the individual student member obtains rewards (i.e., decrease in frustration and

anxiety, decrease in time wasted, increase in success in running a program) which lead to need-satisfaction, and those rewards are recognized by the individual as being obtainable only by acquiring the attitudes of the group. This method of attitude manipulation may have more success than

> a direct attack on the nondesirable evaluation that would be effective only to the extent that associated concepts are changed or that the evaluation reflected by existing . . . concepts is altered. (Shaw & Wright, p. 132).

According to Kelman (1958), there are three theories of attitude change within a group environment:

1. Compliance. This occurs when the individual adopts induced behavior to produce a favorable reaction from the group (i.e., to receive rewards, approval; to avoid punishment, disapproval).

2. Identification. This occurs when the individual accepts group influences for behavior in a desire to establish or maintain a satisfying relationship to the group.

3. Internalization. This occurs when the individual accepts group influences (the ideas and actions of which they are composed) because they are intrinsically rewarding.

Kelman labels the first two types of attitude changing of the individual in the group as social adjustment. However the success of any group technique to manipulate attitudes of persons is contingent upon the

person wanting to belong to the group and perceiving self as being a member of the group.

Attitudes and Learning In Groups

Assessment of student attitude toward a classroom subject has new-found importance as attitudes are being linked to success in learning.

> The development of positive attitudes toward a subject assists the student both in mastering the cognitive aspects of the course materials and in pursuing application and further study of the materials after a course has ended. (Fletcher, 1958, p. 861)

Stanford and Roark (1974) report that groups have not been the rule in traditional schooling practices because of the concept of the

> teacher as an information dispenser and students as information receptacles. This approach has also been commonly emphasizing individual accomplishments, and de-emphasizing cooperative endeavors. The result has been a highly competitive system which emphasizes individual rather than group attainment. (p. 548)

Carl Weinberg (1966) states that in the traditional classroom learning stituation, competitiveness tends to hamper the learning process by promoting a spirit of indifference and resentment in students. The emotional effects of competition raise the general level of anxiety and interfere with problem-solving abilities.

The detrimental effect of competition is supported in studies by Exline (1963) in which goal achievement, problem-solving ability, aggressive behavior and interpersonal communications, all of which are relevant

to learning, were affected.

Since attitudes have been shown to have an affiliative dimension, guiding the formation of interpersonal ties deriving from group membership, examining attitudes of students specific to their group involvement presents a new concern for the educator who employs cooperative processes, rather than competitive ones, for classroom learning.

Johnson and Johnson (1978) state there is consistent evidence from preschool to graduate settings that when cooperative learning groups are emphasized, more positive attitudes are promoted toward subject area and instructional experiences.

One of the major influences on attitudes toward learning reported by the studies conducted by Gunderson and Johnson (1980) is the relationships students develop with peers.

> Within the group, students receive support for the perception that they can be successful if they try to achieve and so successful cooperative learning groups those in which supportive and caring peer relationships develop have been determined to result in higher achievement and more posiitive attitudes toward subject areas. (p. 43)

Agreeing with these findings, Schmuck and Schmuck (1971) report that a student's perception of holding low status (more than the fact of actually having such status) is related to incomplete use of intellectual abilities and to possessing negative attitudes toward the self and toward the school.

An experiment in eliminating all major competitive aspects in a graduate management science classroom was conducted by Goodman and Crouch (1978). Relative performance played no part in the grading scheme and cooperation in the learning experience was rewarded in terms of grades. To establish a cooperative learning experience groups of four were organized in which the students were directed to teach and learn from each other.

A portion of the grade was allocated to peer evaluation and self-evaluation in respect to the student's own assessment of his/her contribution to others' learning.

The investigators noted that student behavior changed gradually and "more and more cooperative learning became evident" (p. 133). The top students were still identifiable but in this cooperative environment, they spent considerable time helping others. "In this manner they were able to raise their own grades and also clarify their own knowledge" (p. 133).

Student comments, in general, indicated that the cooperative learning environment had made the learning experience more profitable (i.e., they had learned more) and more "enjoyable." Goodman and Crouch concluded that the non-competitive aspects of the course were found to have a greater impact on the enjoyment of learning than on the amount learned. This suggests that the course

structure was "successful" in reducing the anxiety produced in similar environments with traditional class structure.

Sharan (1980) compares and evaluates experimental studies employing cooperative small-group learning in the classroom in terms of their differential effects on academic achievement and student attitudes. The first method presented is the "jigsaw classroom" used by Aronson (1978) in which the material to be learned was divided into as many parts as there were members in the group. Each member was responsible for a part and for teaching his/her part to the other group members. Increased self-esteem, choosing to help rather than outdo classmates, was noted by the investigator.

In another study discussed by Sharon, "jigsaw" groups are employed by Geffner (1978). It was found that students maintained positive attitudes toward themselves in terms of their academic abilities and general self-esteem, toward school and toward their classmates. However a decline in attitude toward themselves and toward classmates was noted in students in the control classes, which were taught in the traditional manner.

Replacing interpersonal competition in classrooms with between-group competition, De Vries and Edward (Note 3) performed an experiment using "teams-games-tournaments" (TGT) and Slavin (1978) conducted a study employing "student teams and academic

divisions" (STAD). Utilizing peer-tutoring, the heterogeneous groups appeared, in both studies, to arouse student motivation, but neither study could substaniate any promotion of achievement by the peer-tutoring aspect. However, more positive attitudes toward specific subject matter being studied and greater satisfaction were noted.

The next two approaches presented by Sharan employed the group-investigative approach rather than the peer-tutoring method. Results of an experiment conducted by Sharan and Lazarowitz (Note 4) confirmed the hypothesis that students in small-group classrooms succeed on more high level questions and questions "requiring original elaboration of one's responses than did their peers from traditional classes" (p. 251).

Johnson and Johnson (1975) also used the cooperative learning approach in small investigative groups and report that measures of the social-affective domain showed that students from experimental classrooms were more cooperative and altruistic and much less competitive and selfish. Furthermore, the Johnsons ascertained that students' cooperative behavior skills transferred to their interaction with peers, not members of same learning team, and to their behavior in social situations not structured by the teacher.

Besides reporting superior academic achievement, the Johnsons' results from the cooperative learning method confirm repeatedly that cooperative learning promotes

interpersonal liking, attraction, trust, sense of being accepted by teachers and peers and more positive attitudes toward school and toward the learning situation than does either competitive or individualistic instruction. Students claimed that by participating in cooperative teams "they believed they were learning better in school than did students who studied individually" (p. 253).

In addition to the affective results reported by all methods of small group learning, Sharan states that the helping behavior structured and stimulated by the group investigative methods tends to promote a high level of cognitive functioning and the

> inter-personal exchange within the group appears to foster the emergence of superior problem-solving strategies for all participants regardless of ability level. (p. 256)

The classroom studies of Davidson, Agreen and Davis (1978), Artzt (1979), Goldberg (Note 2), King (1978), Shelly and Cashman (1977), Lemos (1978), Davidson (1974), McKuen and Davidson (1975) involving group process, cited in this chapter and in the following chapter, list positive attitudinal changes in the students toward subject and learning. Moreover, several of these studies found those results to be accompanied by increases in student cognitive achievement.

CHAPTER III

REVIEW OF LITERATURE RELATED TO INSTRUCTIONAL METHODS IN COMPUTER SCIENCE EDUCATION

Methods Of Instruction Not Employing Group Programming

Students study computer science for many different reasons. Some expect to earn a living by working in some phase of computing. Some want to broaden their perspective by learning something about computers. Others have been brought into the introductory computer science class as partial fulfillment of their major's requirements.

Often all these students are placed into the same introductory course (Friedman & Koffman, 1977; Lemos, 1978) creating problems of how to teach to such a wide selection of students and how to construct a curriculum to appeal to such a diversity of backgrounds, needs, motivations and abilities (Singhania, 1980).

The traditional instructional method, dictated by student enrollment and college resources, and widely employed in introductory computer programming language learning courses, is the lecture.

> Numerous theories and methods have been developed for improving educational efforts in computer science classes, but unfortunately, many instructors justify the use of standard lectures on the basis that it is all that is available when in fact, it is only all they want to make available. (Born, Note 5, p. 3)

In the lecture method, students are required to sit passively as the instructor presents theories, facts and constructs, demonstrates their use and leads the students in applications or designs implementing the new theories, facts and constructs. Some lecture courses permit spontaneous questioning by students at any point during the lecture; others, because of large attendance, have a question period after the presentation of the material or refer students to teaching assistants at the conclusion of the presentation. These methods have caused displeasure at all levels: among administrators, instructors, teaching assistants, and especially students.

Questioning whether the traditional lecture is a particularly effective or economical way of teaching programming, Conway (1974) produced the equivalence of lectures on tape. Students were able to use the tapes on cassettes accompanied by 35 mm slides. The initial preparation of each tape required a "tremendous amount of work" (p. 8); however, the completed tapes were non-consumable.

Noting little change in student performance, Conway concluded that the project was essentially a failure because the system was not "portable" and the students, although able to reuse the tapes as often as desired, found the process of searching out particular areas of

misunderstanding to be difficult and slow. In an attempt to correct this complaint, a set of notes was produced that covered all aspects of programming in addition to the rules of syntax.

Conway now views the tape and slide units as essentially remedial and offers beginning computer programming students a two-lecture per week course with optional graduate teaching assistant meetings every afternoon.

Combining the lecture with a laboratory session was also the method used by Unger (1976) and by Prather and Schlesinger (1978) in response to the dissatisfaction expressed by many students and educators with the results of conventional teaching methods.

This solution offered large lectures (250-300 students) with small laboratory sections (20-40 students) which related lecture material to specific disciplines. Both experiments report that a large amount of coordination was needed between lecturers and the graduate teaching assistants in the various laboratory sections in order for the program to function efficiently.

No cognitive or effective results were offered by Unger or by Prather and Schlesinger but student reaction was determined to be favorable except for a common complaint about the tremendous amount of work required. Prather and Schlesinger concluded that the students

seemed better prepared for later courses but the experimenters do not describe how this was determined.

In an attempt to reduce faculty use, Eccles and Gordon (1976) created video tapes of TV lectures that were coordinated with a text in presenting a nonself-paced PL/I (Programming Language I) course to beginning programming students.

This method enabled the educational facility to overcome the lack of computer faculty on regional campuses by permitting students to talk with an instructor, who was present at each showing, via a "talkback system." These students, off the main campus, spontaneously formed small groups in which they helped each other while vying for grades.

The main problems occurred with those students who couldn't tolerate a television approach, became inhibited by using the talkback system or needed more help, but were extremely passive.

No statistics were presented to determine if students did better using this treatment as compared to the "live" classroom, but Eccles and Gordon found the results were acceptable and that the students were exposed to an excellent educational experience.

Stating that students seem to learn less in a traditional lecture setting than may be expected, Daly, Embley and Nagy (1979), designed a "lectureless" environment for 100 business students to familiarize them

with FORTRAN.

In this comparative study, the students in courses with eliminated lectures were given a detailed "programmed" text editor, SIMPLE, that helped them interact with the computer. SIMPLE was a line-oriented text editor with 15 commands and a simple file structure that was independent of hardware and operating systems.

Users were able to create, modify, execute files, and communicate with instructors and each other via files. SIMPLE was designed to overcome the shortcomings of other general-purpose editors: too many commands, ambigious error messages, absence of a HELP feature. Since this course was not self-paced, student progress was determined weekly by ten minute oral quizzes. The instructors found that in addition to initial apprehension and troublesome start-up conditions, teaching load was higher because of the individual oral testing. The solution to that problem was to have the testing procedure delegated to the teaching assistants.

Overall results demonstrate that in the treatment described as "lectureless," students learned as much, if not more, than students in traditional lecture classes. The measure was based on midterm and final examination scores. The "lectureless" students were exposed to more worked out examples and talked more often with members of the class, teaching assistants and instructors than students in the control (lecture) sections.

Besides this, the experimental program offered flexibility in applications and in specifying assignments and could be given with uniformity of instruction, alleviating the need for highly skilled lecturers.

The completion rate of those in the experimental classes appeared to be comparable to conventional (lecture) courses and most students responded favorably when informally asked if they liked the lectureless method.

Another recent approach to introductory computer programming study employs the computer as a teacher, providing a one-on-one tutorial enviroment. Previously, this method, Computer Assisted Instruction (CAI), was deemed unsuccessful in teaching programming because compilers were unable to provide comments on students problem-solving techniques and merely listed the syntactical errors found by a diagnostic system.

PLATO IV CAI, an automatic instructional system exposes introductory students to structured programming concepts and top-down problem solution techniques[1] by means of examples. Students develop on-line, interactive

---

[1]Top-down design is a disciplined approach to organizing complexity. It is similar to writing an outline for a term paper in which the main topic is broken down into a hierarchy of subtopics. These are further broken down until the writer achieves an understandable, controllable amount of complexity. From this level the writer builds, in a step-by-step manner, the solution to the original problem.

dialogue with the tutor (the computer) and the tutor provides hints and comments on structure and efficiency.

Although this is an attractive idea, "intutitively the number of possible solutions must increase enormously as the complexity of the problem increases" (Danielson & Nievergelt, 1975, p. 53) and achieve complexity beyond the sophistication of the system to analyze all the acceptable solution programs.

In another attempt to maximize learning utilizing the computer as the teacher, Gillett (1976) employed the Interactive Program Advising System (IPAS) for beginning programming students. The system aimed at increasing program readability (e.g., by replacing A=2*B by A=B+B) rather than efficiency.

IPAS, unlike a teaching assistant or consultant, did not improve on the student's approach to programming but merely commented on the code the student wrote. However, also unlike a teaching assistant or consultant, IPAS was not able to comment on program logic.

The proponents of this approach to computer education feel a CAI course is at least as effective as the traditional method and although demanding "an enormous time requirement initially of the instructor" (Sjoerdsman, 1976, p. 16), it often reduces the time required by students to master the material. Additionally, this method makes the course flexible and thereby available to many who would not otherwise be able

to enroll.

Employing such a method, Aiken (1981) found an increase in student enrollment and a reduction in staff required to teach a computer course. Aiken states that CAI makes repeated teaching of introductory level courses more tolerable for instructors and more effective for students. Although some students found the method impersonal and even dehumanizing, complaining of eye strain when using the terminal, others liked the constant friendly temperament of PLATO, which kept their mistakes private and was able to repeat lessons without embarrassment.

Aiken admits that the machine "is never as flexible or as responsive as the best human instructor" (p. 82) but recognizes that learning one-on-one from a highly skilled human tutor is not a viable alternative.

Another approach employed in introductory computer courses is one in which the students move at their own speed through the course. Linder (1976) developed an interactive self-paced competency-based course in elementary FORTRAN. Students reacted enthusiastically to the computer-tutor and Linder found this to be an effective method of overcoming the difficulty presented by having to teach students of varying backgrounds.

Ettinger, Goodman and Plumm (1981) developed a combined self-paced, mastery-based FORTRAN course to provide more consistent and predictable learning outcomes

for introductory level courses. It was designed to improve faculty and student satisfaction and was able to decrease the number of redundant FORTRAN courses offered for non-majors and thereby reduce the number of faculty needed to meet the stated needs of the various institutional disciplines.

A constant, insurmountable problem encountered by the method designers was the inability of students to progress due to a lack of adequate strategies for debugging and testing of programs. Ettinger, Goodman and Plum concluded that:

> Just as the traditional lecture-oriented class does not meet the needs of all the students, self-paced, self-instructional courses are not to everyone's liking. (p. 72)

A recent pedagogical method was devised by Dersham (1981) in which students were able to chose 6 of 15 modules in lecture and language to form a course in introductory computer science.

This multipath approach, although difficult to manage and complicating the prerequisite requirements for upper level courses, offered flexibility in use of faculty from other departments, individualization for students of varying abilities, experiences and interests and significant improvement of the standard course offered in previous semesters.

The average student's performance improved and the attitude of student and faculty toward the course became

more positive. Dersham views this model approach as the vehicle for presenting introductory computer science courses in small colleges.

Although computer science educators are employing many innovative methods in introductory computer programming learning courses, taped lectures, TV lectures, slide presentations, lecture-labs, lectures with on-line screen display (Cheng, 1976), lectureless, self-paced, CAI--interactive tutorials (Tsai & Pohl, 1977), masterpiece program readings (Kimura, 1979), varying lecture and language module combinations, none appear to document substantially the consistent improvement of the student, cognitively or affectively.

The Group Programming Instructional Method

The recommendations of the Committee of Undergraduate Curriculum in Computer Science, the consensus of a large number of education and industry professionals, are that:

> Students should be able to read and evaluate programs written by others as well as experience evaluation of their own work and that students should have practice in working alone and also in being part of a programming team and be required to present oral explanation of their particular routine. (Little, 1977, p. 13)

In recognizing the benefits that can be gained from these activities, the committee overtly supports the instructional method employing "egoless" programming within group. Support for "egoless" programming within

the group for the professional as well as for the student, is to be found in Weinberg's (1971) The Psychology of Computer Programming. Weinberg states that success in programming has less to do with intelligence than with personality, work habits and training.

> These things, unlike intelligence, can be changed by experience later in life, which turns the problem from one of selecting programmers to creating them. In other words, good programmers are made, not born. Therefore we should turn our attention to the manufacturing process, or training process. (p. 176)

Research psychologist Sackman (1970), studying patterns of behavior in problem-solving with a computer, agrees with Weinberg that in the end, for programming, experience and training come to dominate all other variables in programming success.

According to Weinberg, placing computer students in programming groups controls their environment, structures their training, dictates their experiences and acts as a pedagogical attempt to achieve such success.

Benefits of Employing Group Programming in Introductory Classes

Weinberg claims there are three main aspects of the process of programming within a group that are designated to benefit the computer science student as well as the professional: helping and/or soliciting help from other members of the group, reading and critiquing other members' programs, and debugging programs written by

group members.

1. Helping or soliciting help from other students in the class is often considered "cheating" (Bezanson, 1975). Working within a group this exchange of information and assistance is encouraged and expected. It combats somewhat the familiar disadvantage of timesharing, in which the students frequently rely too heavily on the computer to find their errors and do not plan ahead carefully (Jehn, Rine & Sondak, 1978; Singhania, 1980).

Weinberg states that when the group members offer and request aid in problem solution and program design and code, the level of programming competency of the group, as a whole, should be raised.

2. Reading and critiquing other members' program solutions for style and clarity, the students see a wide range of work with which to compare theirs (Alford, Hsai, & Perry, 1977; Cashman & Mein, 1975; Conway, 1974; Freeman, 1976; Gries, 1974; Irby, 1977; Khailany, 1977a). Within Weinberg's "egoless" programming teams, members are required to read each other's programs. This removes the guilt felt by students when they read classmates' programs surreptitiously (Lemos, 1978). As they are reading the programs written by others, students pass knowledge about style, language, algorithms, and design to each other (Mackey & Fosdick, 1979).

At the same time as one is reading a group member's

program, one is required also to critique it for good programming style (Alford, Hsia & Perry, 1977; Chi, Morah & Tausner, 1974; Irby, 1977; Khailany & Holland, 1975; Mize, 1976; Perry & Weymouth, 1975; Senn, 1974). The emphasis in computer education today is for "good" programs, not merely correct ones (i.e., those that process data correctly). "Good" programs employ all the techniques of structured programming: top-down design (see Footnote 1), frequent use of comment statements, indentation to indicate refinement, readability, expandability and understandability (Bezanson, 1975; Miller & Petersen, 1981; Ogdin, 1972).

Kernighan and Plauger of Bell Laboratories (1974) strongly support the practice of critical reading as a prelude to learning how to write better programs in the first place. By having to understand programs written by others and by writing code for others to read, students become aware of a necessity of clarity in their programs. Students learn the value of detailed documentation and structured format and are more conscientious in all phases of a programming project. It is found that this clarity in program writing aids the reader and the writer in locating errors in programs that do not run or run incorrectly (Freeman, 1976; Lane, 1975; Nievergelt, 1974; Roth, 1975; Tam & Busenberg, 1977).

3. Debugging programs written by members is the third shared activity within the group. As programs are

read by team members they are not only scrutinized for programming style but also for syntactical and logical bugs. Members also perform walkthroughs[2] with different data sets to help locate logical problem areas for team members. This results in a reduction of individual debugging and editing time since programs get reviewed by team members before they are run on the computer.

Additional consequences of these activities are more efficient use of faculty time, since students now have others to turn to for help, and of academic facilities, since the majority of errors are caught before any terminal time is used. This in turn reduces, and in some cases, eliminates, student frustration, tension and anxiety (Newman, 1973; Shelly & Cashman, 1977), consequently producing an increase in self-esteem and self-competency and encouraging a positive attitude toward computers and computer programming for the beginning student.

---

[2]A walkthrough is the process by which members of the programming group review the design of the algorithm. They "walk through" the input and processing to look for errors in logic and syntax and criticize various aspects of the program design (Bohl, 1982).

Procedural Concerns When Employing Group Programming In Introductory Courses

The use of group programming in the classroom effects practical concern in the following areas: faculty time requirement, student time requirement, grading procedures, group formation and size.

1. One of the main deterrents, cited by educators, to applying group process in introductory computer programming courses, is the amount of faculty time required for success. Teamwork may be new and unnatural for both the student and the instructor and may prove exhausting for both (Crenshaw, 1978; Perry & Sondak, 1978).

Although the commitment required in preparing, developing and especially in maintaining contact with group members is considerably more than in a traditional programming class, Irby (1977) views programming within groups as an excellent alternative to the classical lecture approach when classes are kept small.

Monitoring the action and progress of each group, as well as the participation and progress of the individual student in the class, is a vital facet of the instructor's job (Guha, Carr & Smith, 1977). The instructor's role becomes that of project manager (Baker, 1970; Irby, 1977), mediator, consultant (Menninga, 1974; Williams, 1976) continually encouraging and counseling group members throughout the term (Homeyer, 1977;

Khailany & Saxon, 1978; Korfhage & Smith, 1974; Lemos, 1978; Moccido, 1978).

This considerable increase in workload for faculty, especially for evaluation, is somewhat offset by the easing of the daily work accomplished by getting students to answer many of their own questions or those of their group members and by having to relate to a few groups instead of to a couple of dozen students (Artzt, 1979, Shelly & Cashman, 1977).

2. Heavy time commitments are required not only from instructors but from students as well. Although a major complaint of students in any introductory computer programming course is that the course is demanding of their time, more so than the typical three-credit course, this is especially true in the course that employs group process. As lectures are turned into group discussions of programs or projects, students are forced out of their passive roles into active ones. Now they must communicate orally as well as on paper and time to meet with group members during class as well as outside of the classroom must be arranged (Mavaddat, 1976; Menninga, 1974, Tam & Busenberg, 1977; Weaver, 1978, Williams, 1976).

The success of the programming within a group approach is dependent upon the involvement of all the students, and, with each member being required to critique programs as well as write them, specific

schedules must be followed strictly. Procrastination must become a thing of the past in order for the group to function (Teague, 1981) and it is expected that the interdependence of the group will bring peer pressure to keep deadlines (Freeman, 1976).

However with many students having job and/or home responsibilities, team meetings scheduled in addition to the class time may become extremely difficult or even impossible (Crenshaw, 1978; Freeman, 1976; Homeyer, 1977). Some students may resent spending time outside of the classroom and may not be willing to invest the time demanded for the group project (Buck & Shneiderman, 1976; Costello & Schonberger, 1977, Khailany & Saxon, 1978; Ripley, 1975).

The time spent on informal evaluations of each other's programs and outside of classroom meetings is somewhat offset by the saving of time normally spent on discovering syntactical, logical and spelling errors and by the reduction in the number of runs required to produce a correct program (Shelly & Cashman, 1977; Weinberg, Note 6).

3. Besides the major complaint of the time requirements by the instructor and student, group process demands special grading techniques. Grading a student's program is, in general, a difficult task. Problem-solving is an individual, personal activity and one cannot expect all ideas to emerge in a rigid,

disciplined fashion; but the grading of a program often may be the only source of direct contact between instructor and student (Gries, 1974).

Today simply getting programs to work by any means whatsoever is no longer acceptable. Obscure programs are penalized and requirements of consistent statements, documentation, visible structure and design are upheld.

Reflecting this, computer science educators have grading procedures that vary from allotting 50% for design and documentation and 50% for processing (Williams, 1976) to 20% for processing and 80% for design to 80% for processing and 20% for design (Hazen, Note 7).

When group process is employed in the programming class, the task of grading becomes even more difficult. Now the instructor must be able to evaluate the performance of each student working in the cooperative group milieu (Guha, Carr & Smith, 1977). It becomes necessary to keep written records and have frequent interactions to keep clear the relative strengths and weaknesses of each student (Arnow, 1981; Freeman, 1976, Spence & Groin, 1978).

Some instructors give all students in the group the same grade for the project they complete (Freeman, 1976; Schulman, 1977). Some prefer instructor and teammates to be involved in evaluation. This grading method necessitates peer reviews, written critiques and evaluations of team members' programs (Kenworthy &

Redish, 1979; Khailany & Saxon, 1978).

Other instructors may specify the following grading procedure; 15% for oral presentation, 70% for written presentation and 15% for team performance. Leadership is rewarded and "passengers" are penalized. How the latter is determined was not specified by the practioners but may become obvious in individually taken oral and written exams.

4. Another concern reported by those employing group process in computer science education is how the groups should be created. Groups in programming courses are generally formed using one of three procedures: random selection, self-selection or heterogeneous selection done by the instructor.

In upper level computer courses where the ability and interest of the students is somewhat homogeneous, the method of random selection is often used. But in lower level courses this random approach may form a group consisting entirely of weak students and it offers the possibility of placing students with strongly conflicting personalities into the same group.

Some instructors do random grouping for each assignment. In this manner, if groups with personality conflicts or members of low ability are formed, it is only for a short period of time (Crenshaw, 1978; Lemos, 1978).

To avoid these problems other computer educators

permit students to form their own groups. In this method friends often choose each other and although personality clashes may be avoided, the problem of students of low ability being left out or all placed in one group may occur (Homeyer, 1977; Perry & Weymouth, 1975).

Additionally, in a study done by Korfage and Smith (1974) self-selection led to a poorly functioning group when a strong intimate power struggle developed between two friends on the same team.

A variation of the self-selective method used in upper level courses is to present the class with a list of suggested projects (or subroutines) and have students select the one that interest them most. Those students signing up for the same project (or subroutine) form a group (Cook, 1977; Irby, 1977).

However the method of group formation receiving most favorable report is the one in which the instructor balances the members' abilities to form a heterogeneous group. Sometimes that balance is based on how the students performed on a programming assignment when they were ungrouped (Crenshaw, 1978); sometimes the grade point average or previous experience in computers or group leadership is considered (Comaa, Kramer, & Penney, 1978; Homeyer, 1977; Perry & Weymouth, 1975; Schulman, 1977; Williams, 1976).

The size of groups varies as much as the method of formation. Everything from a "two-member casual joining"

to a declared seven member team is considered a group in computer science programming. Some instructors prefer groups of two or three students (Cooper & Lane, 1976; Freeman, 1976; Kenworthy & Redish, 1979; Korfhage & Smith, 1974; Lemos, 1978; Plum & Weinberg, 1974; Ruschitzka, 1977; Schulman, 1977); whereas others claim that four to five to even seven member groups are the optimum (Comaa et al., 1978; Crenshaw, 1978; Moccido, 1978; Tam & Busenberg, 1977; Williams, 1976).

Freeman (1976) states that the optimal group size is two or three. In larger groups there is difficulty in scheduling weekly meetings and too much opportunity exists for a student member not to carry a fair share of the load. It seems that a balance between a two-member group, where a personal relationship may be distracting and a four-member team where some students could be left out or leave themselves out, is a three-member group.

Three Recent Studies Evaluating The Group Programming Instructional Method

Study I

Employing three-member groups, Basili and Reiter (1981) developed a process metrics that substaniated the effectiveness of disciplined team procedures in an elective computer course for advanced undergraduates and graduates. The comparative experiment involved three-person disciplined groups (DT), three-person ad-hoc

groups (AT) and ad-hoc individuals (AI); all involved in software development.

Although a reasonable degree of homogenity was assumed to exist among personal factors of participants, the experimenters identified two uncontrolled variables to be personal ability and experience and amount of actual time/effort devoted to the project. These were inferred to vary in a random manner among the groups and with information from a pretest questionnaire, an attempt was made to balance the ability/experience of the DT group.

The teams worked independently and the computer activities of each were unobtrusively and automatically monitored by a specially instrumented compiler as the software projects developed.

Basili and Reiter were concerned with both the process used and the product produced by each of the groups. The "heart" of the disciplined team approach was the formal walkthrough (see Footnote 2) required of all its members and the reported results strongly supported the effectiveness of this disciplined methodology in building reliable software efficiently (i.e., fewer bugs, fewer revisions). Moderately substaniated by the experiment was the claim that the product produced by the DT closely resembled that of the AI and was no worse than that of the AT.

Study II

Claiming that every programming class, regardless of whether it is an introductory class for non-majors or the final programming class for majors, should include formal peer review, Shelly and Cashman (1977) designed experiments to study the ability of three-member groups, both of beginners and advanced students, to improve the productivity and quality of programmer software.

Although students learn from making errors and from language diagnostics and incorrect output from their programs, Shelly and Cashman objected to the idea that in data processing failure is considered a natural occurrence.

The experiments demonstrate that peer review of program design and coding caught errors early and conserved time and avoided frustration caused by errors being found during actual program runs. Being aware that their programs would be reviewed, students

> took more time in preparing the programs and were more conscious of whether their program was right or wrong then they would be in the case of their merely coding the program and "throwing" it on the computer to see what would happen. (p. 12)

The students were divided into three-person groups, but the programming was not a group project. Each student completed the design of the program individually. The designs were reviewed but not compared by group members. The walkthroughs (see Footnote 2) insured the group member that his/her program was valid and logical

and would solve the problem. With confidence, the student then converted the design to code and the coded program was then reviewed in a similar fashion for good programming style, syntactical and logical code errors before it was run. If the design was determined incorrect, the student was directed to redesign the solution and arrangements to reconvene as a reviewing group had to be made.

Instructors were available as "chief programmers" (Baker, 1971) and consultants and monitored the performance of each team to identify non-participating students. Each student submitted the number of errors detected, number of runs necessary and the program for the instructors to evaluate. Students received both an individual grade and a team grade determined by the success of the execution of all programs for a given team.

Shelly and Cashman, aware of the problems with conducting structured walkthroughs in an academic environment, report that personality conflicts, differences in ability, non-participating students, improper use of the design and code of participating members by other members of the group (i.e., the copying of programs), could be somewhat controlled by changing group members for each assignment.

However, the problem of absences on the day of the walkthroughs (see Footnote 2) could not be solved by the

experimenters. The three-member group then functioned as a pair but the absentee needed review from the group outside of normal classtime.

The investigators found that students in the study produced a better product and spent more time in educationally meaningful activities such as learning more about program design and good programming code.

> Although the introductory student may not be well versed in a given language, three students on a team have the distinct advantage that what one doesn't know, one of the others probably does. (p. 15)

Study III

In an attempt to establish empirically the "alleged" effectiveness of formal peer review and team debugging techniques in program language learning, Lemos (1979a) conducted a comparative study in an introductory COBOL course involving 215 undergraduate business majors.

Lemos randomly grouped students into four-member teams to read and critique team members program listings and codes. The group review session were scheduled by the instructor in a well-defined regular manner. The teams were reformed after the first run to discuss any undetected errors found by the computer generated diagnostics. The final successful run of the program was the responsibility of the individual students. For each assignment different four-member teams were formed.

The study determined that the experimental groups

received signficantly higher program writing scores on their final exams, and used significantly fewer runs to complete the assignments. No difference was found in the amount of homework completed by both the experimental and control classes.

Lemos concluded that classroom lectures can be more effectively supported by structured walkthroughs (see Footnote 2) conducted within groups and that although the results were promising, further empirical studies involving team activities in learning a programming language were needed.

CHAPTER IV

PROCEDURES OF THE STUDY

The Setting

This study was conducted at a private, coeducational, suburban four-year college during the Spring 1982 semester, and involved three classes of an introductory level course, entitled Structured Programming Using FORTRAN. The educational institution services approximately 10,000 part-time and full-time commuter students under a policy of open enrollment, offering certificate, associate, bachelor and master level degrees, within day, evening and weekend course schedules. The college is recognized as a leader among those institutions responding to the "new" student on campus (see Appendix A).

Structured Programming Using FORTRAN is designed as the second course in the computer science sequence at the institution. It is intended to substanially develop the student's ability to employ good programming techiques such as structured format, top-down design (see Footnote 1) and extensive documentation while introducing the student to the high level language, FORTRAN. The students are taught to solve problems by forming

algorithms using a structured pseudocode[3] and to translate them into FORTRAN code, to be executed on a DEC 20 PDP/11 minicomputer.

Selection of the Experimental and Control Classes

All three classes met during the day, one at the main campus on Monday and Wednesday from 9 a.m. to 10:50 a.m., and two at the branch campus, one on Saturday from 9 a.m. to 11:50 a.m. and the other on Thursday from noon to 2:50 p.m. Both campuses draw the majority of their students from the Westchester suburbs.

Although the two classes at the branch campus would appear to be more similar (meeting on same campus, once a week), it has been found (observed by investigator and confirmed by colleagues) that weekend and evening students are usually employed full-time, older, non-matriculated and often taking one course for self-enlightenment. This was substaniated by the student questionnaire filled out by each student in the study during the first class meeting.

Consequently the two weekday classes were selected for the major part of the investigation--the comparison of the student working within a fixed programming group

---

[3]Pseudocode is an informal design language that is used to write algorithmic structured problem solutions. The solution is then translated into a formal programming language.

to the programming student working individually, within the traditional lecture method. Random selection assigned the experimental treatment to the branch campus afternoon class and the control to the main campus class. The physical separation of the classes further insured the segregation of the teaching methods being examined in the investigation.

Group programming was employed also in the Saturday morning class but the students were placed in different groups for each programming assignment. Having the main campus class as the control placed both classes employing programming groups on the branch campus where students could meet and discuss the programs from the course in the open fashion they were being directed to do in their classes.

Determination of Homogenity Of Classes

In order to substaniate any observed experimental results as comparable, the diversity of the college student body necessitated assessing the backgrounds of all the subjects in the study. Each student was required to complete a student questionnaire (see Appendix B), on which the student listed his/her sex, age, number of work-hours per week, number of college credits completed, number of computer courses completed, number of credits being carried for the semester.

In addition, the student's scores on the college

placement examinations covering arithmetic, algebraic and verbal skills were included in the assessment (see Appendices C, D and E). The college requires all matriculated students to be given placement exams; however, many subjects in the study were either transfer or certificate students or just were taking one course and as such had not taken any placement examinations. These students were given the examinations by the investigator during the first few weeks of the term, under the conditions prescribed by the Basic Skills Coordinator, and at times convenient to both examiner and examinee.

The arithmetic, algebraic and English usage placement examinations are multiple choice and were graded by the investigator. A fourth component of the placement procedure is the writing of an essay defending a position taken by the student with respect to a situation presented. Three English instructors are needed to grade this examination. Since these services were unavailable during the semester, this part of the placement procedure was not considered for pretreatment evaluation.

In order to complete an accurate assessment, it was necessary to ascertain the pretreatment programming ability of the students. The prerequisite for Structured Programming Using FORTRAN is a course called Introduction to Programming, which combines data processing with BASIC

language programming. The investigator has taught that course several times and has co-authored the programming texts used by the students (Farley, Grossman & Tucciarone, 1979; Grossman & Tucciarone, 1981). However many of the students in Structured Programming Using FORTRAN do not take that required course because they have had a similar course(s) elsewhere (perhaps in a different programming language).

To measure the pretreatment programming skill of all the students the investigator designed a short (45 minutes) Preliminary Programming Proficiency Test (PPP). The PPP (see Appendix F) is language independent and tests those aspects of programming skill that a student would be expected to acquire in a first exposure to a programming language.

The investigator had conducted pilot studies in Spring 1981 and in Fall 1981, and had administered the PPP test to six classes of Structured Programming Using FORTRAN. It was found to assess adequately the basic programming skills of students entering that course.

The PPP is divided into three major parts: debugging, reading and writing.

1. Debugging. Students are given grammatical rules of a BASIC-type language and are required to detect syntactical errors in lines of code.

2. Reading. Students are required to demonstrate the understanding of a given algorithm, presented as a

flowchart, in detailed multi-step levels, by determining the output given specific sets of input.

3. Writing. Students are required to perform four separate modifications of an algorithm presented in flowchart form and as a coded BASIC program. (The modifications may be done in BASIC code or as drawings on the flowchart.)

Specifically tested are the abilities to use assignment statements, initialize a variable, establish cumulating values, employ looping, search and compare data, branch conditionally, manipulate character data and handle simple data structures.

Students may have strengths or weaknesses in any one or all three parts of the PPP. A student who may not have strong analytic expertise may be a skillful debugger and so do well on Part 1 and poorly on Part 3. Since the three aspects of programming skill are all important in the process of programming (Lemos, 1977; Shneiderman, 1980), strength in an one of the them would be a positive contribution to a programming group.

To get a general programming performance profile of each subject the PPP score was the sum achieved on all three parts but the specific strengths and weaknesses were noted. Although the PPP was not able to accurately rank the students, it was able to discriminate between the two extremes of programming performance.

The experiment did not begin until the third

programming assignment and so the grades on programming assignments #1 and #2, and the writing time, terminal time and number of runs required for programming assignment #2 were also included in the student comparability assessment.

Eighteen items were used for the student comparability assessment. Their codes are discussed below:

1. Sex. Male students were given a code of 0 and female students a code of 1.

2. Age. Students of traditional college age, seventeen to twenty-two, were given a code of 1; students whose ages were above twenty-two were given a code of 2.

3. Educational Status. The educational status of the student was based on the level of formal education that the student had completed: freshman level or below received a code of 1, sophomore level, a code of 2, junior level, a code of 3, senior level, a code of 4, college graduate level, a code of 5, and graduate school level, a code of 6.

4. Semester Load. The semester load was the actual number of college credits being taken by the student for the semester in which the experiment was conducted. Minimum semester load was three credits.

5. Computer Background. The computer background of the student was the number of computer course credits completed by the student. Minimum computer background

required for the course in which the experiment was being conducted was three credits.

6. Number of Work-hours Per Week. The number of hours per week the student was employed in a paying job was used for this item.

7. College Placement Exam in Arithmetic. A code of 0 was used to indicate that the student placed below average level; a code of 1 indicated that the student placed at the average level; and a code of 2 indicated that the student placed above the average level in arithmetic ability.

8. College Placement Exam in Algebra. This item was coded similarly to Item 7.

9. College Placement Exam in Usage. This item was coded similarly to Item 7.

10. PPP Part 1. The score achieved by students on Part 1 of the PPP reflected the student's ability to locate and correct errors in programming code. Minimum score was zero and maximum score was ten.

11. PPP Part 2. The score achieved by the student on Part 2 of the PPP reflected the student's ability to read a computer program. Score range was from zero to ten.

12. PPP Part 3. The score achieved by the student on Part 3 of the PPP reflected the student's ability to modify an existing computer program. Score range was from zero to ten.

13. Total PPP Score. Score range for this item was from zero to thirty and was the sum of the scores obtained by the student on Items 10, 11 and 12.

14. Grade of Programming Assignment #1. The first assignment essentially directs the student through all the steps involved in creating and editing a program file and is given as the first assignment in all Structured Programming Using FORTRAN courses. Score range was from zero to ten.

15. Grade on Programming Assignment #2. The second assignment requires the student to write a simple program using good programming style and reflects the material in the first two chapters of the textbook used in the course. Score range was zero to ten.

16. Writing Time. This score represented the time (in hours) the student spent writing, designing and correcting the program solution for assignment #2. The time was recorded by the student.

17. Computer Terminal Time. This score represented the time (in hours) spent by the student at the computer terminal for programming assignment #2. The time was recorded by the computer and displayed for the student at logout.

18. Number of Runs. This item was the number of runs needed for a correct execution of programming assignment #2 and was recorded by the computer as the generation number of the student's file.

The 18-item student assessment for homogenity of classes was analyzed by *t*-test statistics to identify any items demonstrating significant differences in the three classes. Any items so identified were considered when interpretating the results of Comparisons 1, 2, and 3.

Tables 1 presents the results of the analysis of the 18-item assessment of homogenity for the control class employing the traditional method of individual programming and for the experimental class employing fixed student programming groups. Table 2 presents the results of the assessment for the control class and the class in which the students were programming in randomly assigned groups. Table 3 presents the results for the two classes employing the experimental instructional methods.

Table 1

Item Mean and Standard Deviation on the Student Comparability Assessment for Comparison 1

	Control Class (Individual Programming)		Experimental Class (Fixed Group Programming)	
Item	M	SD	M	SD
1	.4762	.512	.5500	.510
2	1.3333	.483	1.4000	.503
3	2.0952	1.411	2.9500	1.932
4	12.5258	3.558	8.4500	5.094*
5	4.0000	1.449	4.9500	3.120
6	13.2857	13.990	17.9000	18.474
7	.9524	.218	1.0000	.000
8	1.0000	.000	.9500	.224
9	1.1429	.573	1.2000	.523
10	6.9524	1.830	6.4500	1.395
11	7.7143	2.004	7.5500	2.685
12	4.1190	3.053	4.5250	2.688
13	18.7857	3.700	18.5250	4.959
14	7.7619	3.315	9.2900	1.891*
15	7.7286	3.150	8.4000	3.222
16	1.6500	1.504	1.6944	1.077
17	1.5556	1.228	1.2944	.747
18	5.3333	5.018	2.8333	3.451

*$p < .05$

Table 2

Item Mean and Standard Deviation on the Student Comparability Assessment for Comparison 2

Item	Control Class (Individual Programming) M	SD	Experimental Class (Changing Group Programming) M	SD
1	.4762	.512	.5789	.507
2	1.3333	.483	1.7368	.452*
3	2.0952	1.411	2.5263	1.954
4	12.5258	3.558	6.7895	4.467*
5	4.0000	1.449	5.0526	2.838
6	13.2857	13.990	31.0526	11.525*
7	.9524	.218	.9474	.229
8	1.0000	.000	.8947	.315
9	1.1429	.573	1.2632	.452
10	6.9524	1.830	6.4211	2.036
11	7.7143	2.004	7.0526	2.758
12	4.1190	3.053	3.7368	3.186
13	18.7857	3.700	17.2105	6.877
14	7.7619	3.315	9.0526	1.877
15	7.7286	3.150	8.1842	3.185
16	1.6500	1.504	1.7474	1.992
17	1.5556	1.228	1.5737	1.234
18	5.3333	5.018	4.0000	4.761

*$p < .05$

Table 3

Item Mean and Standard Deviation on the Student Comparability Assessment for Comparison 3

Item	Experimental Class (Fixed Group Programming) M	SD	Experimental Class (Changing Group Programming) M	SD
1	.5500	.510	.5789	.507
2	1.4000	.503	1.7368	.452
3	2.9500	1.932	2.5263	1.954
4	8.4500	5.094	6.7895	4.467
5	4.9500	3.120	5.0526	2.838
6	17.9000	18.474	31.0526	11.525*
7	1.0000	.000	.9474	.229
8	.9500	.224	.8947	.315
9	1.2000	.523	1.2632	.452
10	6.4500	1.395	6.4211	2.036
11	7.5500	2.685	7.0526	2.758
12	4.5250	2.688	3.7368	3.186
13	18.5250	4.959	17.2105	6.877
14	9.2900	1.891	9.0526	1.877
15	8.4000	3.222	8.1842	3.185
16	1.6944	1.077	1.7474	1.992
17	1.2944	.747	1.5737	1.234
18	2.8333	3.451	4.0000	4.761

*$p < .05$

Formation of Groups In The Experimental Classes

Considering the computer background of the students and their PFP score, the students were divided into three categories--"crackerjack" programmers, "weak" programmers, and "between-the-extremes" programmers. The specific score at which one category ended and another began was dependent on the particular scores in that class and where the appropriate divisions had to be made to insure that the each category would have the same number of students. This procedure was employed in an attempt to maximize the possibility of forming heterogeneous three-person groups in which one student from each category would be selected.

In the class in which the heterogeneous groups would be fixed for the entire semester, the investigator balanced the programming ability (PFP score)/experience (recorded on the Student Information Sheet) of the participants and formed three-person groups (Basili & Reiter, 1981; Shneiderman, 1980).

In the class that was to experience group programming within changing groups, a computer program (see Appendix G), written by the investigator, randomly generated different heterogeneous three-person groups (i.e., employing the method of stratified random selection) for each programming assignment. Coincidently

both branch campus classes initially had student bodies that were multiples of three. However as the term progressed student attrition necessitated combining some groups and permitting groups of two students. Care was given so the reformed groups maintained the intended balance wherever possible.

The Course Outline

All three daytime classes were taught by the investigator and covered the same material, from the same texts and lecture notes and at the same pace. One section was directed to program individually, another section was directed to program within assigned fixed heterogeneous groups and the last section was directed to program within randomly assigned heterogeneous groups that would be changed for every programming assignment. The three classes shall be referred to, respectively, as IND, FHG and RHG.

A course outline was distributed to all sections at the first class meeting. It detailed the objectives and the pacing of the course, the material covered in each lecture, the readings from the texts, the schedule of assigned programs and grade evaluation procedures.

The IND's outline included the familiar edict--"Do your own work. Any programs that are shared will cause both students involved to share a single grade" (see Appendix H). The outline of the FHG and RHG classes

replaced that with a few paragraphs about team programming, peer review and structured walkthroughs (see Appendix I). To foster group interaction and, more specifically, cooperation, students were told that their programming assignment grade would be an average of the grades received by all the members of their team for that assignment. Their course outline also included a listing of what particular stage of the assigned program would be due in class each week.

The Group Programming Process

The group programming process was described to the FHG and RHG sections as three, twenty-minute class meetings per assigned program. The first would be for problem analysis, and include sketching of an algorithm (flowchart) and writing of a pseudocode solution (see Footnote 3).

The second session would be an exchange of their FORTRAN coded program solutions, hand written or listed by the computer. Each student would peruse his/her team member's program for syntactical errors. Data would be supplied with each assignment so the students could walk through each program in search of logical bugs. Up to this point no machine-student (i.e., computer-program) interaction was required.

After the noted bugs were removed, the group would meet in class for a third time to go over the first

computer run of each team member's program. During this session the group would discuss the errors found by the computer (the compiler diagnostics) that caused a program not to run or the incorrect results of a program that did run. Students would note the errors that the group review had missed.

The third group session for an assignment overlapped with the first group session for the next assignment. During that class meeting the groups would meet to finish reviewing each other's program run, and then would listen to a lecture. Then the same group, in FHG, and a new group, in RHG, would meet to analyze the next assignment. Questions on the lecture dealing with new material could be asked of group members as well as of the instructor.

The final correct code and run of a programming assignment was the responsibility of each student to complete and hand in the next week. Meetings with members of the group outside of the class were encouraged. For many students with family and/or job commitments this was often impossible.

To facilitate the exchange of work between members in a group, the campus terminal room contained a box into which the members were to deposit and/or pickup a copy of the completed code, listing or run of a group member's program, due to be discussed in the group review session that week. This procedure supported the free, open atmosphere proposed in class in which there was to

be no secret coding and it was "okay" to see programs of others and receive help. If the "drop-off" box were used by the team, each member would be able to review the other member's programs before the class met again. Of course, lack of time, always a limiting factor, prevented many students from doing assignments early and participating in the "drop-off" box.

All three classes did seven assigned programs that became progressively more difficult. Distribution of exemplary solutions was done in an effort to cure the problem of assignments being handed in late. Deadlines had to be adhered to in order to insure that students would keep pace with the course and be able to participate in the group process and so the penalty for late assignments was a 50% reduction in program assignment grade (Fosdick & Mackey, 1979).

> Left to their own devices, students will postpone working on a program they do not fully understand and then attempt to do it in a burst of energy near the due date. (Noonan, 1979, p. 188)

The classroom time spent by the FHG and RHG in group work was used by the IND for individual analysis, design and desk checking[4] of programming assignments, going over homework and doing classroom problems and programs.

---

[4]A desk check is a visual inspection of the program, tracing the path of sample data through the program, to eliminate errors in syntax and logic before submitting the program to the computer (Sanders, 1979).

In all sections the instructor walked around the room and acted as as consultant and resource person--giving hints when necessary. Any questions that came up frequently, indicating a source of misunderstanding, in any of the three classes were discussed in front of the entire class.

Data Collection by Students

Consideration was given to two devices that may be used to record anecdotal and self-reported data from the students in the study. One method is to have the student maintain a log book during the semester, recording various facts about each programming assignment and hand in the log book at the end of the course. The other method is to have the students hand in the requested data after each programming assignment.

The first method was used during the pilot studies conducted in 1981. Each student was requested to complete a log book, recording the total time spent writing and rewriting each program, the number of runs needed to produce a correct program, the total time spent in the terminal room and any comments, pro or con, about the specific program assignment or, for those in the grouped classes, about the group programming experience. Even though the students were constantly reminded to keep their log books up to date it proved to be too much of a chore for them. At the end of the semester most student

log books were empty, inadequately filled out or hastily finished. This device proved inefficient.

Consequently the second device, which required students to hand in a summary sheet with each completed programming assignment, was employed in this study and proved to more effective in obtaining the requested data from the students.

On the Assigned Program Summary Sheet (see Appendix J) the students recorded:

1. The number and type (syntactical or logical) of errors found in the program before it would run correctly. (The assumption here is that eventually all programs handed in run successfully; however, often the output is incorrect or the program has not considered certain data that would cause errors if they had been used.)

2. The time to analyze the problem, write, and rewrite the coded solution. (For the RHG and FHG this included the time spent in group sessions in and out of class.)

3. The number of runs of the program before it "worked." (This was recorded by the computer and appeared on the printout as the generation number of the file. For example, EGRO.FOR.28, indicates 28 revisions of the file EGRO.FOR.)

4. The time spent on the terminal to get a completed assignment. (Included would be the time used to create a

file, type in a program, edit the program, execute the program and then, if necessary, edit and execute the program again until it "ran" successfully. This was recorded by the computer and printed after the student logged off the system. All the terminal use time for that assignment would be totaled by the student and recorded on the summary sheet.)

The summary data sheet method of student data collection proved more successful in keeping all the students up to date in their data collection than the student log book method.

In addition to the summary sheet, the grouped classes were required to rate the effectiveness of the group walkthrough for that programming assignment (see Appendix K); and to insure that they read each other's programs, do evaluation ratings on the programs written by their team members (see Appendix L). Both of these ratings used a seven-point scale and were developed and used by Shneiderman (1980). The students participating in the group programming experience in the pilot studies of 1981 were asked to use and critique the review ratings. Any questions found to be ambigious, insignificant or inapplicable to the classroom situation were eliminated and the resulting ratings were used in the present study.

The investigator discussed the meaning and intent of each question (on both rating sheets with the RHG and FHG

classes) prior to the first review in an effort to decrease the chance of different interpretations of the questions and the rating scale (Shneiderman, 1980).

Grading Procedures

Programming assignments were graded on a scale of zero to ten. Each programming assignment was compared to a model solution written by the instructor. Since there is no one correct solution (Lemos, 1977), and the efficiency of the program measured by the amount of CPU used during program execution (also printed at logout by the computer) and the number of lines of code were not considered, programs that processed the sample data and produced correct results received eight out of ten points. The remaining two points were earned for "good programming style." The consensus among computer programming learning experts is that certain features of style help produce programs that are easier to read, easier to correct and easier to modify.

The basic style components emphasized in Structured Programming Using FORTRAN are (a) overall top-down design (see Footnote 1), (b) clearly worded descriptions of the purpose and the procedure of the program, (c) use of meaningful mnemonic variable names, (d) description of use of each variable in a data dictionary, (e) declaration of all variables, (f) indentation to show structure and grouping, (g) extensive use of comment

statements for anything that is not obvious, and (h) clear descriptions of input format and output display. A quarter of a point was deducted for each feature missing on a program. Each programming assignment reflected approximately one chapter in the course textbook and was similar to the problems at the end of that chapter.

The midterm and final examinations were shown to departmental colleagues who agreed that both tests responded to the objectives listed by the instructor. The format of the midterm and final were similar to that of the PPP except that they were both open-book examinations. The first part of each examination presented lines of code, data names, numeric and string values that the student was required to determine as valid or invalid, and included the interpretation and writing of simple lines of programming code that performed calculations or evaluations.

The second part of each examination presented a program written in pseudocode (see Footnote 3) or in FORTRAN code. The student was required to walk through the program with specific data and determine the output. These first two parts of the midterm and final examinations were graded objectively.

The third part of each examination required the student to write a program in FORTRAN. A model answer, written by the instructor, was reviewed by two colleagues on the computer faculty. Point distribution was

discussed and the version used for the grading procedure was determined.

Programs written by the students in response to the third part of each examination were graded by the two computer science faculty members involved in the solution design and the instructor. The results of the three graders agreed in 88.3% of the cases and the differences, when they occurred, were no more than two points.

In order to maintain examination security, two forms of the midterm and final were used. The IND and FHG classes were given the same examinations but the RHG class received alternate forms. The framework of both examinations were identical and tested the same programming skills (see Appendices M and N).

The students in the control class, IND, that met twice a week took the first and second parts of the midterm and final during one class and the third part during the next class meeting. The students in the experimental classes, FHG and RHG, that met once a week took the midterm and final in their entirety during a single class meeting. Extra time for the examinations (midterm and final) was available for all students.

Attitude Measurement

A primary concern of this investigation, in addition to cognitive gains, is the attitude toward computers promoted by the participatory group process as

it compares to that generated by the traditional instruction format. The most frequently used methods of measuring attitude (Likert, 1932) requires subjects to indicate their agreement or disagreement with statements about the attitude object. Endorsement of the statement serves as the basis for inferring the existence of positive or negative evaluations on the part of the endorser (Shaw & Wright, 1967). The investigator compiled a list of statements about computers and computer programming from a combination of items on the Mathematical Anxiety Rating Survey (Suinn, Note 8), the Minnesota Computer Literacy and Awareness Assessment (Note 9) and similar surveys used in the studies done by Hazen (Note 7) and Lemos (1977).

Each statement offered the student five options for response: strongly disagree, disagree, no opinion, agree, strongly agree. The responses were rated on a scale from one to five or from five to one depending upon whether the statement about computers and computer programming was positively or negatively worded.

The survey was administered to 57 students in the Spring 1981 pilot study, at the beginning and again at the end of their semester in Structured Programming Using FORTRAN. Student names were not requested on the survey. Besides selecting an opinion, the students were asked to comment on the clarity of each statement. Those items that students found ambigious or did not elicit a

definite positive or negative response were reworded or removed.

The revised survey was administer to 28 students in the pilot study of Fall 1981, at the beginning and the end of the semester, and again they were asked to additionally comment as to the clarity of each item. The resultant survey of 18 statements formed the Attitude Toward Programming Survey (see Appendix O) and was given to the IND, RHG and FHG classes at the beginning of the term. Students were asked to indicate their course section but not their names. At the end of the term, before the seventh programming assignment, all classes, again anonymously, retook the same survey to note the movement of a class in a positive or negative direction.

Student Reactions to the Course and the Seventh Programming Assignment

To aid in the interpretation of the results obtained in Comparisons 1, 2 and 3, student reactions to the course were solicitated and a seventh programming assignment was required.

The seventh programming assignment, a team project, was a large program that had to be separated into programming modules. Each member of the team designed and coded one or more modules. The complete solution to the assignment required the linking together of all the modules. Each member of the team received the same

grade.

The students in the IND class were divided into heterogeneous teams (according to the grades they had earned in the course) and were included in this assessment as a comparison to the teams in the other classes that had been programming in groups for the major part of the semester.

In addition, all students in the study were asked to list the most positive and most negative aspects of the course and make selections from opposite word-pair descriptions of the course. It was expected that these open questions and directed selections would produce praise or condemnation for the group programming process.

CHAPTER V

ANALYSIS OF DATA

The following hypotheses and subsequent data are offered in response to the questions posed in the Introduction. Comparisons 1, 2 and 3 examine four hypotheses each. The organization of the twelve hypotheses is presented in the chart below.

	Programming Achievement			Attitude
	Proficiency	Perseverance	Efficiency	
Comparison 1 (IND and FHG classes)	Ho_1	Ho_2	Ho_3	Ho_4
Comparison 2 (IND and RHG classes)	Ho_5	Ho_6	Ho_7	Ho_8
Comparison 3 (FHG and RHG classes)	Ho_9	Ho_{10}	Ho_{11}	Ho_{12}

Comparison 1

Comparison 1 contrasts the programming achievements and attitudes of students receiving instruction in the traditional method of lecture and individual programming (IND) with those of students receiving instruction in the experimental method employing lecture and fixed heterogeneous student programming groups (FHG). Three dimensions of student programming achievement are

discussed. They are programming proficiency, programming perseverance, and programming efficiency.

Hypothesis One

H_0 : The teaching method employing fixed heterogeneous student programming groups in combination with lectures, when compared to the method employing lecture and individual programming, has no differential effect on student programming proficiency in an introductory computer programming course as measured by:

1. the grades of programming assignments #3, #4, #5, #6 (GPA3, GPA4, GPA5, GPA6),

2. the midterm examination score (MIDTERM),

3. the final examination score (FINAL).

Programming assignments are graded on a scale from zero to ten.

The results of analysis by t-test statistics are presented in Table 4. A time series analysis of the grades on programming assignments #1 through #6 is presented in Figure 2.

Table 4

Mean and Standard Deviation of Programming Proficiency Measures Used in Comparison 1

	IND		FHG	
Measure	M	SD	M	SD
GPA3	6.5000	3.142	8.4500	2.620*
GPA4	6.5000	4.177	8.2300	2.818
GPA5	5.5857	4.629	7.2150	3.884
GPA6	5.0619	4.320	7.6500	3.777*
Midterm	80.3333	9.707	77.8000	14.322
Final	64.7619	22.816	67.9000	31.007

*$p < .05$

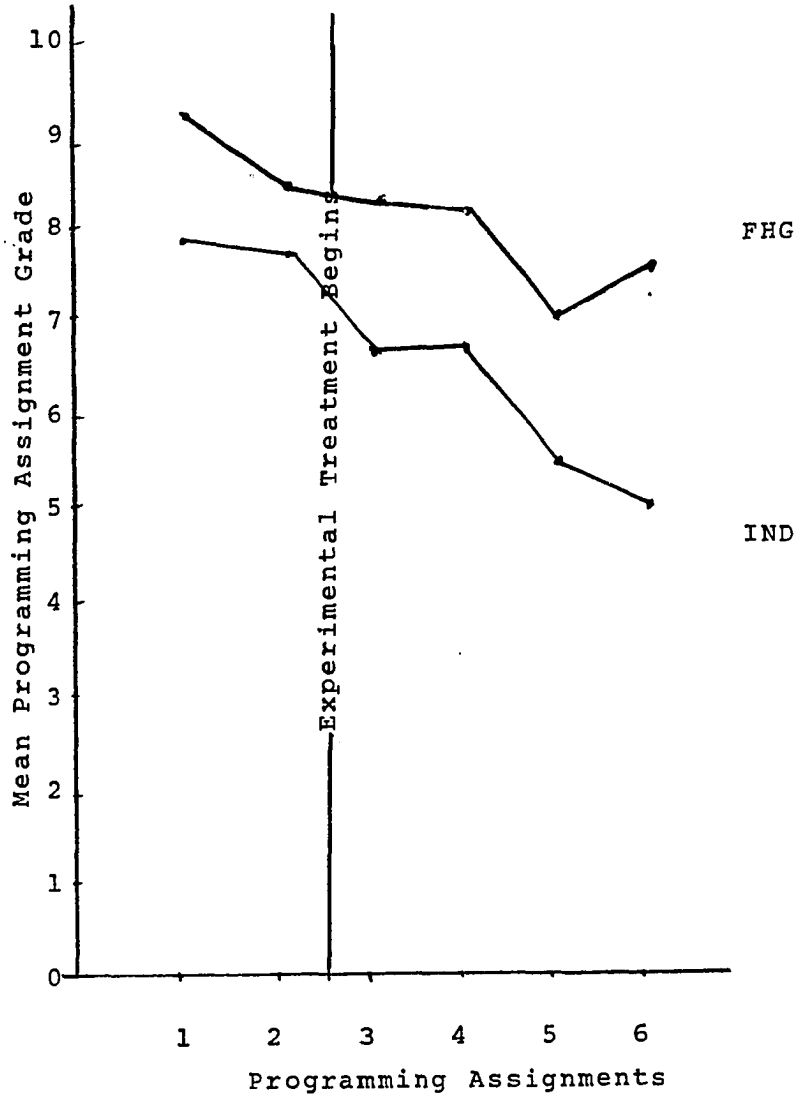


Figure 2. Time series analysis of the mean programming assignment grades in the IND and FHG classes.

The results of analysis by the two-tail t-test determines that the differences in the grades on programming assignments #3 and #6, achieved by the IND and FHG classes, are significant. In both instances the FHG class scores higher than the IND class.

The group programming approach, with its cooperative algorithimic designing and coding sessions, appears to aid its members since the grades on all the programming assignments in the FHG class are higher than those in the IND class. However, the grades achieved by the students in the FHG class on the midterm examination, in comparison with those achieved by the students in the IND class, are lower.

The FHG class earns higher programming grades on the first two assignments done individually, but the difference in the programming grades achieved by the classes increases after the experimental treatment begins. Since the student who does not hand in a program receives a grade of zero and all students in the class are included in the programming grade analysis, this difference in programming grades between the two classes must be considered in conjunction with the number of students in each class who do not complete assignments.

The lack of significant differences in the midterm and final examination scores in the FHG and IND classes causes any differences observed in the programming grades to be considered the results of the group supporting its

members in writing programs rather than the group producing members with better programming skills. Ho_1 must be regarded as defensible since the data offer insufficient evidence to reject it.

Hypothesis Two

Ho_2: The teaching method employing heterogeneous fixed student programming groups in combination with lectures, when compared to the method employing lectures and individual programming, has no differential effect on student programming perseverance in an introductory computer programming course as measured by:

1. the number of students completing programming assignments on time,

2. the number of students completing programming assignments late,

3. the number of students not completing programming asignments.

Table 5 presents the numbers of students in the IND and FHG classes in each category for each programming assignment.

Table 6 presents the mean and standard deviations of the programming perseverance measures for students in Comparison 1. Significant results of analysis by t-test statistics are noted.

Table 5

Number of Students in Comparison 1
(A) Completing the Programming Assignment on Time
(B) Completing the Programming Assignment Late
(C) Not Completing the Programming Assignment

	IND			FHG		
Programming	Category			Category		
Assignment	A	B	C	A	B	C
#1	18	2	6	24	0	2
#2	22	0	4	24	1	1
#3	14	4	3	17	2	1
#4	13	3	5	15	4	1
#5	12	1	8	15	3	2
#6	9	5	7	15	3	2

Table 6

Mean and Standard Deviation of Programming Perseverance Measures Used in Comparison 1

Measures	IND M	IND SD	FHG M	FHG SD
Number of programming assignments completed on time	2.2381	1.480	3.0500	1.605*
Number of programming assignments completed late	.6667	.856	.5000	1.100
Number of programming assignments not completed	1.0952	1.578	.4500	1.146

*$p < .10$

In the IND class, three students fail to hand in assignment #3; five students fail to hand in assignment #4; eight students fail to hand in assignment #5; and, seven students do not hand in assignment #6. The programming assignments #3, #4, #5, and #6 are done in group sessions in the FHG class and every student, but one or two, complete each assignment. In the IND class, the same assignments are done individually and as the term progresses more students do not hand in their programs and receive a grade of zero. Evidence indicates that as the term progresses more students in the FHG class are completing assignments than are students in the IND class.

The data supports the consideration of the instructional method employing fixed heterogeneous student programming groups to have a differential effect on student programming perseverance. Ho_2 is rejected.

Hypothesis Three

Ho_3: The teaching method employing heterogeneous fixed student programming groups in combination with lectures, when compared to the method employing lectures and individual programming, has no differential effect on student programming efficiency in an introductory computer programming course as measured by:

1. the time spent by the student in designing, coding and debugging each programming assignment

(recorded by the student, listed as writing time and measured in hours);

2. the time spent by the student at a computer terminal for each programming assignment (recorded by the computer, listed as terminal time, displayed at logout, and measured in hours);

3. the number of runs required by the student for each programming assignment (recorded by the computer, listed as number of runs and displayed as file generation number).

Any student not handing in the above data for a particular assignment is not included in the analysis for that programming assignment.

Table 7 presents the results of analysis of this data using t-test statistics. Figures 3, 4 and 5 display time series analyses of the programming efficiency measurements.

Table 7

Mean and Standard Deviation of Programming Efficiency Measures Used in Comparison 1

Measure	IND M	IND SD	FHG M	FHG SD
Programming Assignment #3				
Writing time	1.9643	.981	1.5062	.670
Terminal time	2.0154	.884	1.6437	.614
Number of runs	7.3571	4.181	4.0000	4.775*
Programming Assignment #4				
Writing time	2.1917	.956	1.9000	1.264
Terminal time	2.6182	1.582	2.3937	.920
Number of runs	10.0000	6.841	7.1875	7.250
Programming Assignment #5				
Writing time	2.8667	1.224	3.2000	1.461
Terminal time	2.7000	2.179	2.8733	1.242
Number of runs	12.4444	11.226	9.6000	9.448
Programming Assignment #6				
Writing time	2.4375	1.146	2.7000	1.935
Terminal time	2.1000	.856	3.0200	1.168*
Number of runs	8.6250	4.033	7.4000	4.485

Note. Writing time and Terminal time are measured in hours.

*$p < .05$

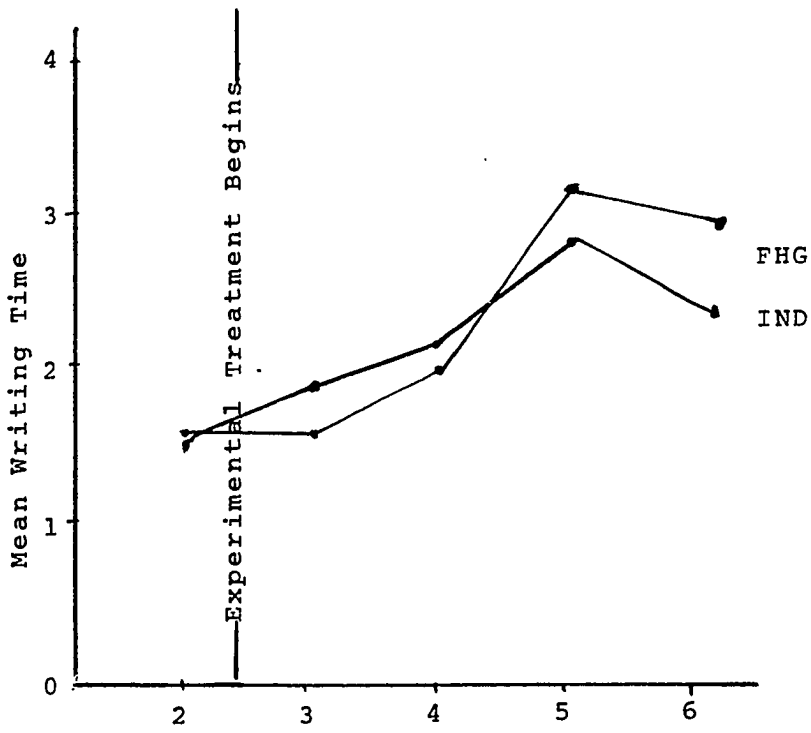


Figure 3. Time series analysis of the mean writing time (in hours) per programming assignment in the IND and FHG classes.

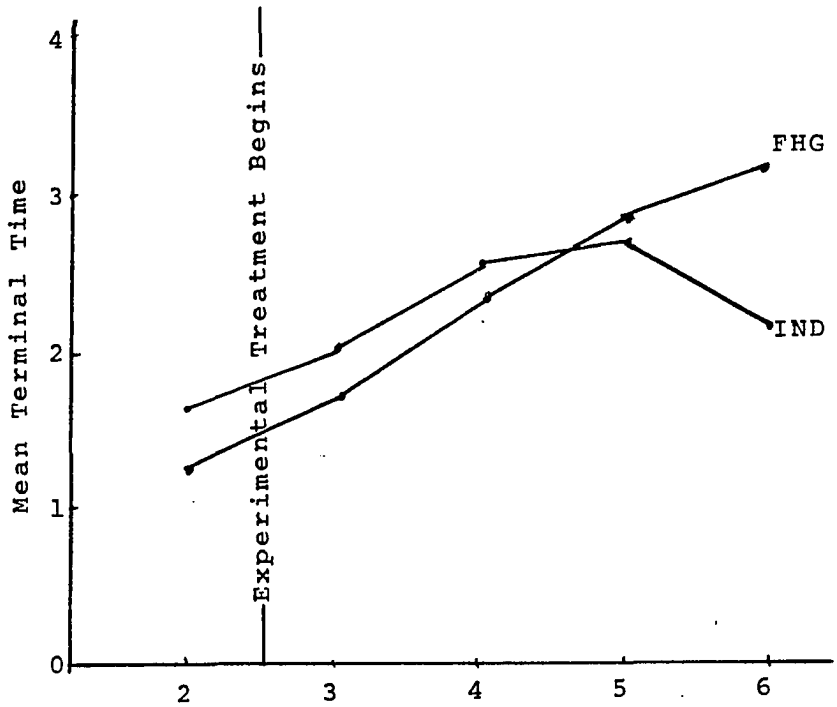


Figure 4. Time series analysis of the mean terminal time (in hours) per programming assignment in the IND and FHG classes.

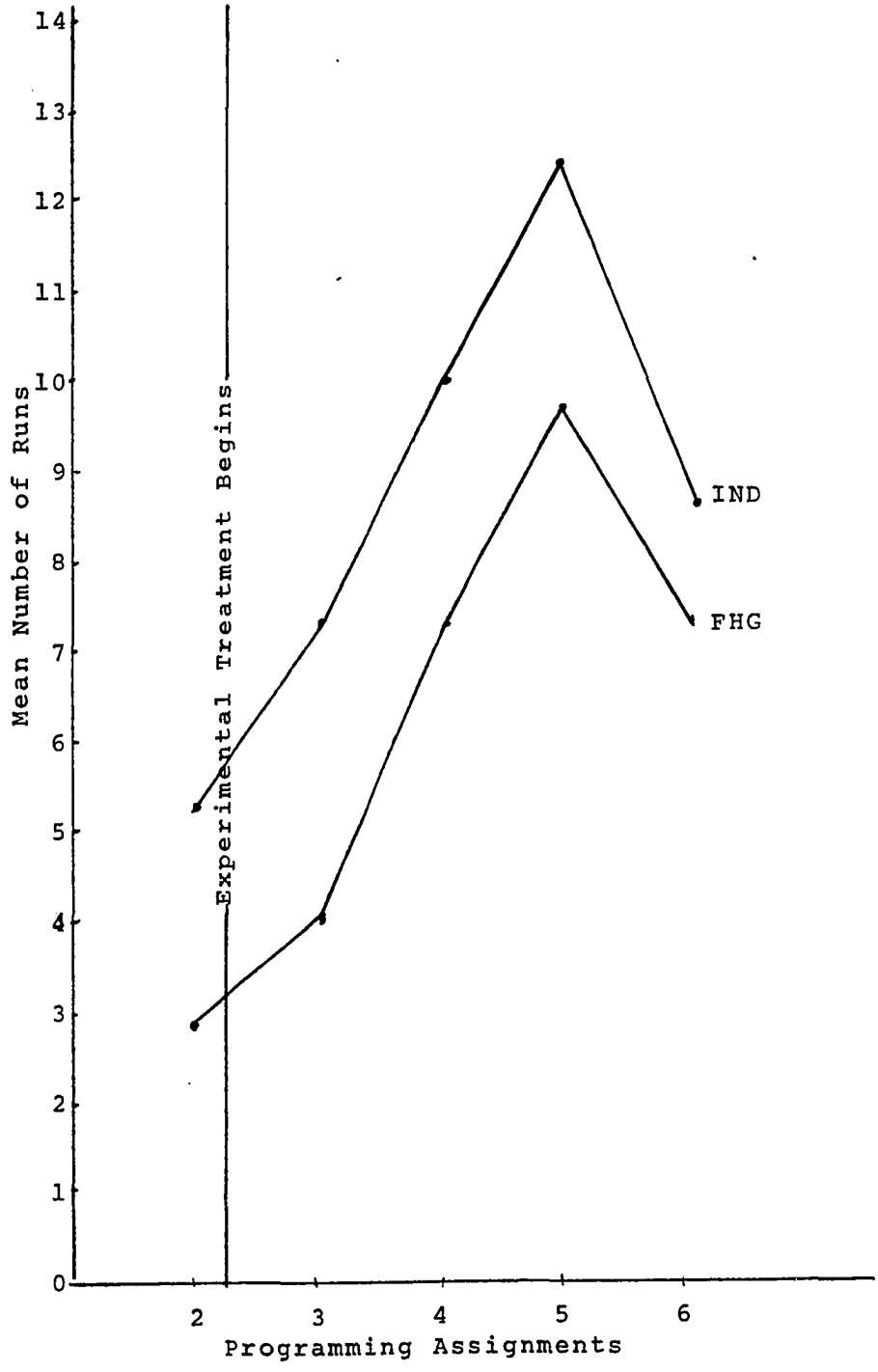


Figure 5. Time series analysis of the mean number of runs per programming assignment in the IND and FHG classes.

Although the IND class has 21 students and the FHG class has 20 students, the number of students who do not report the efficiency measurements is fewer for each programming assignment in the FHG class. These students are excluded from the analysis since a measurement of zero indicates an amount of zero time spent writing a program, working on the program at the terminal or that there are no runs completed on the program. The differences between the two classes are found to be significant in the number of runs required for programming assignment #3 (FHG<IND) and in the terminal time spent for programming assignment #6 (IND<FHG).

As the programs become more difficult both the FHG and the IND classes spend more time writing the programs. However the last programming assignment, #6, reverses that trend in both classes.

In the time series analysis of writing time, displayed in Figure 3, the FHG class initially is spending less writing time per assignment than the IND class; but by the end of the experiment, the students in the FHG class are spending more writing time per assignment than the IND class. The time being spent by the student in analyzing and designing an algorithm, writing code and debugging (writing time) appears to increase when working within the group milieu.

This may be the result of the student being more aware of the time spent on this writing process since it

is done in groups in and outside of the class.

The second measure of programming efficency used for this hypothesis is the time spent on the terminal by the student for each programming assignment. Evidence indicates that the group approach appears to have caused increased time to be spent on the terminals as the programs become more difficult. Both classes display increases of time spent on the terminals for the third, fourth and fifth programming assignment, and by the fifth programming asssignment, the students in the FHG class are spending more time at the terminals than the IND students.

The largest difference in the terminal time being spent by the students in the IND and FHG classes is for the sixth programming assignment. The number of students reporting terminal time for that assignment in the IND class is eight and each student receives a perfect grade on the assignment. Better programming students may require less terminal time than weaker programming students.

The third measure of programming efficiency is the average number of runs per program needed to obtain correct output. The number of runs is lower for all the programs done by the FHG class; however, only for programming assignment #3 is this difference found to be significant.

The instructional method employing fixed student

programming groups, when compared to the traditional method, does not appear to have a differential effect on programming efficiency. Ho_3 must be regarded as defensible since the data offers insufficient evidence against it.

Hypothesis Four

Ho_4: The teaching method employing fixed heterogeneous student programming groups in combination with lectures, when compared to the method employing lectures and individual student programming, has no differential effect on student attitude toward computers and computer programming in an introductory computer programming course as measured by:

1. the Attitude Toward Computers and Programming Survey given pretreatment,

2. the Attitude Toward Computers and Programming Survey given posttreatment (after the sixth programming assignment).

An analysis by *t*-test statistics is done for each statement on the questionnaire. The results of this analysis appear in Table 8.

Table 8

Pretreatment and Posttreatment Mean Item Response on the Attitude Toward Computers and Computer Programming Survey in Comparison 1

	Pretreatment		Posttreatment	
Item	IND	FHG	IND	FHG
1	4.8750	4.4400 *	4.5455	4.5556●
2	4.2609	4.4400	3.7273	4.3333 *
3	4.5000	4.5200	4.1634	4.3889
4	4.5000	4.1667	3.6818	4.3333●*
5	3.7917	3.8800	3.5909	4.0000●
6	4.5000	4.4000	4.4091	4.1667
7	3.9565	3.8000	3.7273	3.5556
8	4.6250	4.0400	4.1818	4.3889●
9	4.1667	3.8400	3.5909	3.9444●
10	3.5833	3.4800	2.9091	3.6111● *
11	4.2917	4.0800	4.1364	4.3333●
12	4.0833	4.0400	3.7273	4.2778●*
13	4.1250	4.2400	3.8182	4.3333●
14	3.7917	3.7200	3.5455	3.8889●
15	4.0417	4.1200	3.4545	4.2778●*
16	3.3478	3.2800	2.6190	3.2353
17	3.6250	3.7600	3.7273●	3.5556
18	3.7917	3.8800	3.6364	3.9412●

Note. A score of 5 indicates a most favorable attitude toward the item.

● Indicates an increase of item mean response posttreatment compared to pretreatment in same class.

*$p < .05$

The Cronbach α, a measure of reliability, is used to determine the internal consistency of this instrument by correlating each item on the instrument with every other item. The Cronbach α for the Attitude Toward Computers and Programming Survey is 0.84789, indicating high internal consistency. Consequently an analysis of the total response average on the Attitude Toward Computers and Programming Survey conducted in the IND and FHG classes pretreatment and posttreatment is presented in Table 9. Figure 6 displays the same results graphically.

Table 9

Mean and Standard Deviation of Pretreatment and Posttreatment Student Total Response Average on the Attitude Toward Computers and Computer Programming Survey in Comparison 1

Survey	IND		FHG	
Source	M	SD	M	SD
Pretreatment	4.0764	.332	3.9978	.487
Posttreatment	3.7247	.446	4.0401	.545*

*$p < .05$

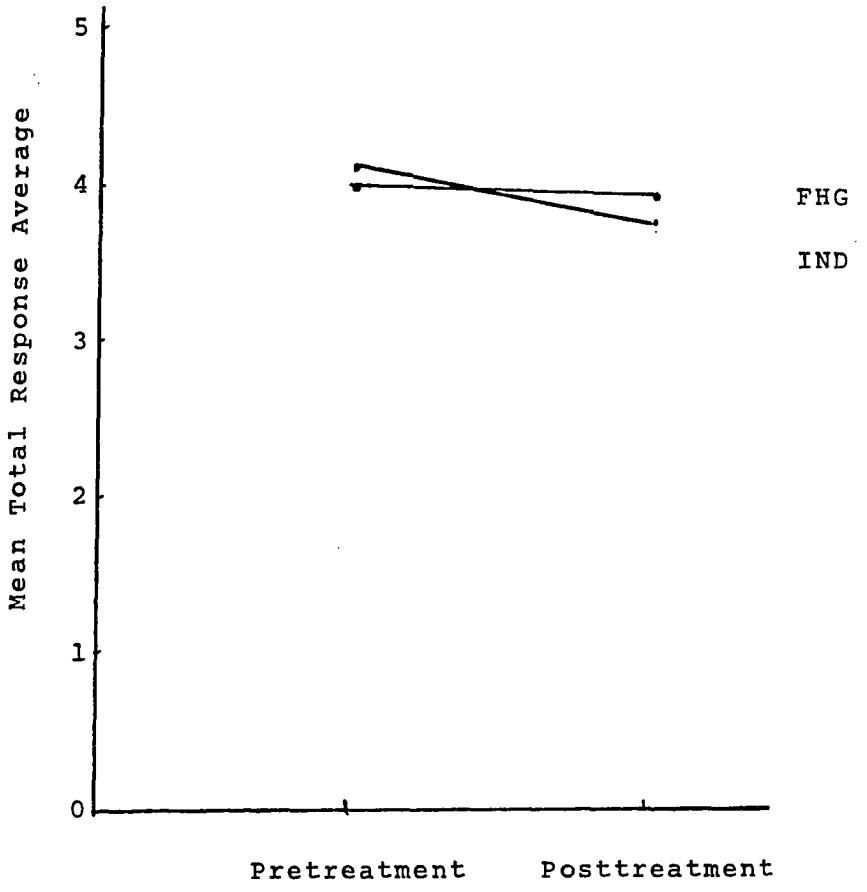


Figure 6. Pretreatment and posttreatment means of student total response average on the Attitude Toward Computers and Computer Programming Survey in the IND and FHG classes.

The results of comparing the pretreatment responses on the 18 statements of the Attitude Toward Computers and Programming Survey produces one statement that elicits a significant difference in responses from the two classes. The IND class scores a more favorable response to the statement: Computer programming is an important skill to have for future job opportunities.

In comparing the posttreatment responses of the two classes to the 18 statements of the Attitude Toward Computers and Programming Survey, five statements are found to produce significant differences in responses. On each of these statements the FHG class scores a more favorable response than the IND class. The five statements are:

I try to avoid working on my computer programs until the last minute. They are my least favorite assignments.

I always look forward to working on my computer programming assignments.

I hesitate working with computers because they are strange and anxiety-provoking.

Computers are fascinating and exciting to work with.

I view programming as a stimulating and challenging acitivity.

In addition to the first four statements listed above the following statements receive more favorable response scores in the FHG class, in the posttreatment survey, compared to the responses given by the class in

the pretreatment survey:

Computer programming is an important skill to have for future job opportunities.

I become frightened and panicky at the thought of having to write a computer program.

Computer programming seems to be a fairly useless skill.

Computers are extremely accurate, efficient and reliable.

Computers are important for the efficient operation of large and small businesses.

I find that discovering solutions to programming problems is a logical process.

Every college student should be required to take a course in computer programming.

One cannot use what is taught in this course outside of a programming environment.

In the pretreatment survey the IND class responds more favorably to 11 out of the 18 statements than does the FHG class. In the posttreatment survey the FHG class responds more favorably to 17 out of 18 statements than does the IND class.

In comparing the total response average of both classes, a significant difference is noted in the posttreatment results in which the FHG class achieves a more positive response score than does the IND class. The data provide sufficient support for a rejection of H_{0_4}.

Comparison 2

Comparison 2 contrasts the programming achievements and attitudes of students receiving instruction in the traditional mode of lecture and individual programming (IND) to those of students receiving experimental instruction employing lecture and variant heterogeneous student programming groups (RHG).

Hypothesis Five

Ho_5: The teaching method employing changing heterogeneous student programming groups in combination with lecture, when compared to the method employing lecture and individual programming, has no differential effect on student programming proficiency in an introductory computer programming course as measured by:

1. the grades on programming assignments #3 (GPA3), #4 (GPA4), #5 (GPA5), #6 (GPA6),
2. the midterm examination score (MIDTERM),
3. the final examination score (FINAL).

Students who do not hand in a programming assignment receive a score of zero. Maximum score on each programming assignment is ten. The midterm and final examination score range is from zero to one hundred.

Analysis of data by t-test statistics is presented in Table 10. Figure 7 displays a time series analysis of

the average grades on the programming assignments for the IND and RHG classes.

Table 10

Mean and Standard Deviation of Programming Proficiency Measures Used in Comparison 2

	IND		RHG	
Measure	M	SD	M	SD
GPA3	6.5000	3.142	8.8842	1.352*
GPA4	6.5000	4.177	8.7632	2.669*
GPA5	5.5857	4.629	7.0421	4.023
GPA6	5.0619	4.320	6.7263	3.778
Midterm	80.3333	9.707	73.7368	23.385
Final	64.7619	22.816	61.2632	28.423

*$p < .05$

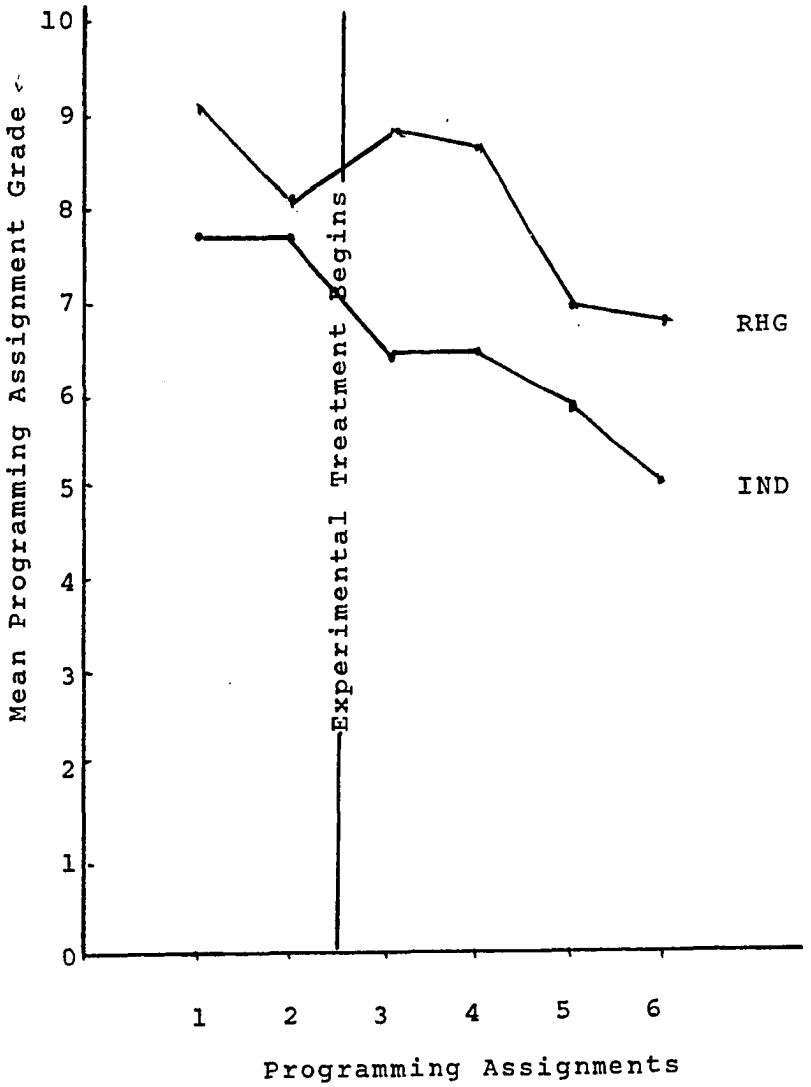


Figure 7. Time series analysis of the mean programming assignment grades in the IND and RHG classes.

All the average programming assignment grades attained by the RHG class are higher than those earned by the IND class; however, only the differences in the class grades for programming assignments #3 and #4 are found to be significant.

The graphs of the average class programming assignment grades, displayed in Figure 7 appear to be parallel. Consideration must be given to the fact that the students who do not hand in the assignments receive a grade of zero and are included in the analysis.

The ability of the RHG class to earn higher grades on the programming assignments, done in groups, is not reflected in the scores achieved by the class on the midterm and final examinations, taken individually.

The lack of any significant differences in the midterm and final examination scores in the IND and RHG classes indicates that any differences that are observed must be considered to be the results of the group supporting its members in the writing of programs rather than in either teaching method producing differential programming skills. Ho_5 must be regarded as defensible since the data offer insufficient evidence to reject it.

Hypothesis Six

HO_6: The teaching method employing variant heterogeneous student programming groups in combination with lectures, when compared to the method employing

lecture and individual programming, has no differential effect on student programming perseverance in an introductory programming course as measured by:

1. the number of students completing the programming assignments on time,

2. the number of students completing the programming assignments late,

3. the number of students not completing the programming assignments.

Table 11 presents the numbers of students in the IND and RHG clases in each category for each assignment.

Table 12 presents the means and standard deviations of the programming perseverance measures for the classes in Comparison 2. Significant results of analysis by t-test statistics are noted.

Table 11

Number of Students in Comparison 2
(A) Completing the Programming Assignment on Time
(B) Completing the Programming Assignment Late
(C) Not Completing the Programming Assignment

	IND			RHG		
Programming Assignment	Category A	B	C	Category A	B	C
#1	18	2	6	17	2	0
#2	22	0	4	17	0	2
#3	14	4	3	18	1	0
#4	13	3	5	16	2	1
#5	12	1	8	14	1	4
#6	9	5	7	14	3	2

Table 12

Mean and Standard Deviation of Programming Perseverance Measures Used in Comparison 2

Measures	IND M	IND SD	RHG M	RHG SD
Number of programming assignments completed on time	2.2381	1.480	3.2632	1.195**
Number of programming assignments completed late	.6667	.856	.3684	.761
Number of programming assignments not completed	1.0952	1.578	.3684	.761*

**$p < .05$
*$p < .10$

The assignments #3, #4, #5 and #6 are done in variant student programming groups in the RHG class. The number of students completing the first programming assignment done within the group is 19 and this number decreases slightly during the term. The number of students completing the sixth programming assignment is fourteen. Fifteen students, who are programming individually in the IND class, complete the third programming assignment an this number decreases substanially so that only nine students complete the sixth programmin assignment on time.

The student count in the IND class is initially 25 and only 21 students finish the course. The RHG class begins and ends the semester with a student count of 19.

A student who does not keep pace with the work in the course (as evidenced by failure to complete assignments) maybe considered as not continuing, much in the same manner as a student who drops the course.

The retention of all the students in the RHG class and the significantly higher student program completion mean and lower student failure to complete the program mean support the rejection of Ho_6.

Hypothesis Seven

Ho_7: The teaching method employing variant heterogeneous student programming groups in combination with lectures, when compared to the method employing

lecture and individual programming, has no differential effect on student programming efficiency in an introductory computer programming course as measured by:

1. the time spent by the student in designing, coding and debugging each programming assignment (reported by the student, listed as writing time and measured in hours);

2. the time spent by the student at the computer terminal for each programming assignment (recorded by the computer, listed as terminal time, displayed at logout, measured in hours);

3. the number of runs required by the student for each programming assignment (recorded by computer, listed as number of runs and displayed as the file generation number).

Students who do not hand in the above data for a program are not included in the analysis for that programming assignment.

Table 13 presents the results of analysis of the data by t-test statistics.

Table 13

Mean and Standard Deviation of Programming Efficiency Measures Used in Comparison 2

Measure	IND M	IND SD	RHG M	RHG SD
Programming Assignment #3				
Writing time	1.9643	.981	2.8444	2.520
Terminal time	2.0154	.884	2.0647	1.521
Number of runs	7.3571	4.181	6.6111	6.204
Programming Assignment #4				
Writing time	2.1917	.956	2.7000	2.349
Terminal time	2.6182	1.582	2.1812	.949
Number of runs	10.0000	6.841	11.0625	7.750
Programming Assignment #5				
Writing time	2.8667	1.224	2.9643	1.599
Terminal time	2.7000	2.179	3.1643	2.023
Number of runs	12.4444	11.226	8.5000	6.959
Programming Assignment #6				
Writing time	2.4375	1.146	2.4333	1.203
Terminal time	2.1000	.856	2.4250	1.218
Number of runs	8.6250	4.033	9.0625	7.733

Note. Writing time and Terminal time are measured in hours.

Although the IND class has 21 students and the RHG class has 19 students, the numbers of students who do not report the efficiency measurements are fewer for each of the programming assignments in the RHG class. The students who do not submit a summary report are eliminated from the analysis for that assignment. Since those students who are reporting data are the better students in the IND class, the efficiency measurements for that class are lower (i.e., better) than they would be if all the students in the class complete the assignments and report the required data.

In addition the students in the programming groups report spending more time writing and analyzing programs but this may be the result of being more aware of time being spent in these activities since they are done in class. The majority of the students in the RHG class work during the weekdays and are not available to meet outside of class and so the writing time measurement esssentially reflects the time spent in class, in groups, and the time spent at home, working alone.

The second measure of programming efficiency used is the average time spent on the terminal by the student for each programming assignment. The data reported indicates no significant differences in the measures achieved by both classes. Since the number of students reporting data is smaller in the IND class than in the RHG class, the IND class average appears to be lower than it might

have been if more students had completed the assignments or had reported the summary data requested.

The third measure of programming efficiency used in the study is the average number of runs per programming assignment needed to obtain correct output. Again no significant difference is obtained between the two clases on this measurement. The class average is affected by any extreme score reported by a student. One student in the IND class records 55 runs for programming assignment #5 and the class average is strongly influenced by this value since only nine students report any results.

None of the programming efficiency measurements obtained attain statistical significance and time series analyses of these measurements produce no recognizable trends. Ho_7 is tenable since the data offer insufficient evidence against it.

Hypothesis Eight

Ho_8: The teaching method employing variant heterogeneous student programming groups in combination with lectures, when compared to the method employing lectures and individual programming, has no differential effect on student attitude toward computers and programming as measured by:

1. the Attitude Toward Computers and Programming Survey given pretreatment,

2. the Attitude Toward Computers and Programming

Survey given posttreatment (after the sixth programming assignment).

Analysis of each statement on the survey, by *t*-test statistics, is presented in Table 14.

The Cronbach α of 0.84789, indicating high internal consistency in the measuring instrument, permits analysis of the average student total response score. The analysis of the average total student response is presented in Table 15 and in Figure 8.

Table 14

Pretreatment and Posttreatment Mean Item Response on the Attitude Toward Computers and Computer Programming Survey in Comparison 2

	Pretreatment		Posttreatment	
Item	IND	RHG	IND	RHG
1	4.8750	4.6842	4.5455	4.7778●
2	4.2609	4.3158	3.7273	4.3889●*
3	4.5000	4.5556	4.1634	4.8889●
4	4.5000	4.3158	3.6818	4.3529●*
5	3.7917	3.9474	3.5909	4.0000●
6	4.5000	4.6316	4.4091	4.5000
7	3.9565	4.0000	3.7273	3.5882
8	4.6250	4.3684	4.1818	4.5556●
9	4.1667	4.0516	3.5909	3.8889
10	3.5833	3.8947	2.9091	3.7222 *
11	4.2917	4.1053	4.1364	4.1667●
12	4.0833	4.2632	3.7273	4.2222
13	4.1250	4.2105	3.8182	4.2222●
14	3.7917	4.0526	3.5455	3.3333
15	4.0417	4.2105	3.4545	4.3889●*
16	3.3478	3.5789	2.6190	3.4444 *
17	3.6250	3.5789	3.7273	3.7222●
18	3.7917	3.9474	3.6364	3.5556

Note. A score of 5 indicates a most favorable attitude toward the item.

●Indicates an increase of item mean response posttreatment compared to pretreatment in same class.

*p .05

Table 15

Mean and Standard Deviation of Pretreatment and Posttreatment Student Total Response Average on the Attitude Toward Computers and Computer Programming Survey in Comparison 2

Survey	IND		RHG	
Source	M	SD	M	SD
Pretreatment	4.0764	.332	4.1374	.440
Posttreatment	3.7247	.446	4.0154	.459*

*$p < .05$

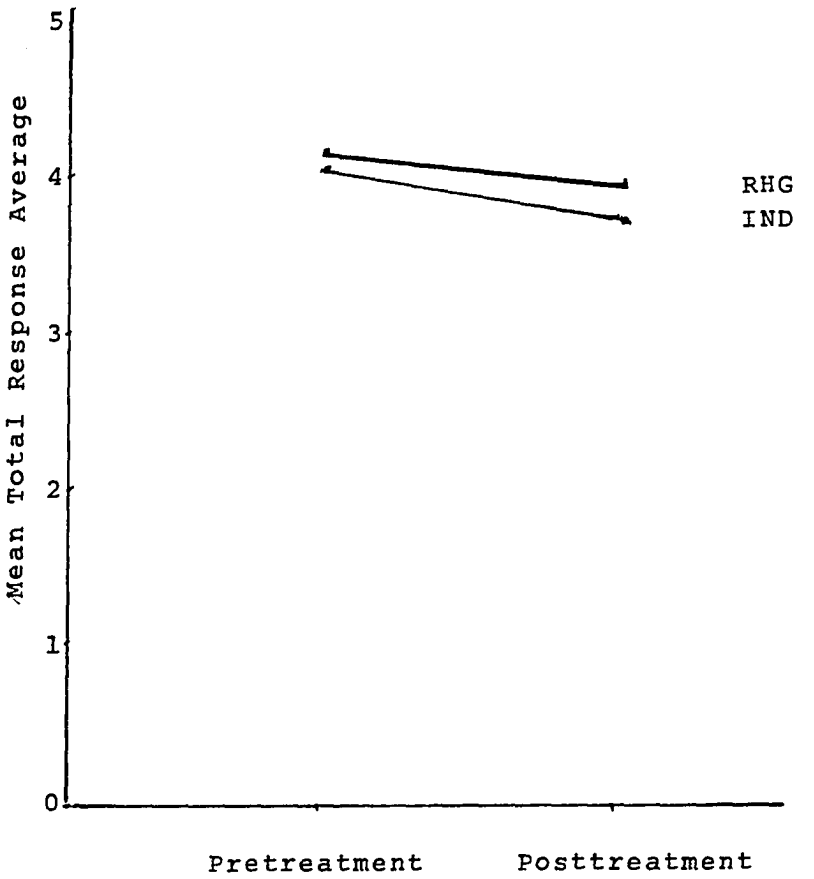


Figure 8. Pretreatment and posttreatment means of student total response average on the Attitude Toward Computers and Computer Programming Survey in the IND and RHG classes.

The results of comparing the pretreatment responses to the 18 statements on the Attitude Toward Computers and Programming Survey produces no statements eliciting significant differences in response from the IND and RHG classes. In comparing the posttreatment responses of the two classes, five statements are found to produce significant differences in responses and in each of these statements the RHG class scores more favorably than the IND class. The five statements are:

I try to avoid working on my computer programs until the last minute. They are my least favorable assignments.

Computers are fascinating and exciting to work with.

I view programming as a stimulating and challenging activity.

I always look forward to working on my computer programming assignments.

I find that trying to get my program to run on the computer is a frustrating ordeal.

In addition to the first three statements listed above, the following statements receive more positive response scores in the RHG class in the posttreatment survey compared to the reponses given by the that class in the pretreatment survey:

Computer programming is an important skill to have for future job opportunities.

Computers are not very useful because they are

always breaking down or making mistakes.

I become frightened and panicky at the thought of having to write a computer program.

Computer programming seems to be a fairly useless skill.

Computers are important for the efficient operation of large and small businesses.

I find that discovering solutions to programming problems is a logical process.

It seems to me that a programming class motivates students to cheat--they copy other people's programs and hand them in.

The IND class scores a more favorable reponse on one out of the 18 statements on the posttreatment survey compared to the responses given by the class on the pretreatment survey. That statement is: It seems to me that a programming class motivates students to cheat--they copy other people's programs and hand them in.

In the pretreatment survey the IND class responds more favorably to 6 of the 18 statements than does the RHG class. In the posttreatment survey, the IND class again responds more favorably to 6 of the 18 statements but only one of these statements in the posttreatment survey is the same as the ones the IND class indicates more favorable response to in the pretreatment survey. That statement is listed above and is the same statement

that receives more favorable response in the IND class posttreatment than pretreatment.

In comparing the total response average of the classes, a significant difference is noted in the posttreatment results. The class experiencing group programming has a significantly more favorable attitude toward computers and computer programming than the class receiving instruction in the traditional mode. The data provide sufficient support for a rejection of Ho_8.

Comparison 3

Comparison 3 contrasts the programming achievements and attitudes of students in the classes receiving instruction in the two different experimental modes (FHG and RHG).

Hypothesis Nine

Ho_9: The teaching method employing fixed heterogeneous student programming groups in combination with lectures, when compared to the method of employing changing heterogeneous student programming groups in combination with lectures, has no differential effect on programming proficiency in an introductory computer programming course as measured by:

1. the grades on programming assignments #3 (GPA3), #4 (GPA4), #5 (GPA5), #6 (GPA6),

2. the midterm examination score (MIDTERM),

3. the final examination score (FINAL).

Students who do not hand in a programming assignment receive a grade of zero. Grades on the programming assignments range from zero to ten. Grades on the midterm and final examinations range from zero to one hundred.

The means and standard deviations of the programming proficiency measures are presented in Table 16.

Table 16

Mean and Standard Deviation of Programming Proficiency Measures Used in Comparison 3

	FHG		RHG	
Measure	M	SD	M	SD
GPA3	8.4500	2.620	8.8842	1.352
GPA4	8.2300	2.818	8.7632	2.669
GPA5	7.2150	3.884	7.0421	4.023
GPA6	7.6500	3.777	6.7263	3.778
Midterm	77.8000	14.322	73.7368	23.385
Final	67.9000	31.007	61.2632	28.423

Analysis by t-test statistics reveals no significant differences in the average programming grades, midterm or final examination scores in the FHG and RHG classes. Although the average grades for programming assignments #3 and #4 are higher for the RHG class, the FHG class scores higher average grades for programming assignments #5 and #6 and on both the midterm and final examinations.

Plausible explanations for the observed differences may lie in the inability of the RHG students, who are working full-time, to meet the increased time requirements of the course as the term progresses and the retention of all the students (including the weak students) in the RHG class.

Since no significant differences are found on the measures of programming proficiency for the students in the classes employing the fixed or the variant programming group, Ho_9 must be considered defensible.

Hypothesis Ten

HO_{10} : The teaching method employing fixed heterogeneous student programming groups in combination with lectures, when compared to the method of employing changing heterogeneous student programming groups in combination with lectures, has no differential effect on student programming perseverance in an introductory computer programming course as measured by:

1. the number of students completing the programming

assignment on time,

2. the number of students completing the programming assignment late,

3. the number of students not completing the programming assignment.

Table 17 presents the numbers of students in the FHG and RHG clases in each category for each assignment.

The means and standard deviations of the programming perseverance measures for the classes in Comparison 3 are presented in Table 18.

Table 17

Number of Students in Comparison 3
(A) Completing the Programming Assignment on Time
(B) Completing the Programming Assignment Late
(C) Not Completing the Programming Assignment

	FHG			RHG		
Programming	Category			Category		
Assignment	A	B	C	A	B	C
#1	24	0	2	17	2	0
#2	24	1	1	17	0	2
#3	17	2	1	18	1	0
#4	15	4	1	16	2	1
#5	15	3	2	14	1	4
#6	15	3	2	14	3	2

Table 18

Mean and Standard Deviation of Programming Perseverance Measures Used in Comparison 3

Measures	FHG M	FHG SD	RHG M	RHG SD
Number of programming assignments completed on time	3.0500	1.605	3.2632	1.195
Number of programming assignments completed late	.5000	1.100	.3684	.761
Number of programming assignments not completed	.4500	1.146	.3684	.761

The measure of student programming perseverance is determined by the number of programming assignments completed by the students in the class. One hundred percent of the students in the FHG class complete programming assignment #1 on time. This percentage decreases for the following assignments and reaches its lowest level of 75% for programming assignment #6. One hundred percent of the students in the RHG class complete the first programming assignment and this percentage also decreases as the term progresses and reaches its lowest level of 73% on programming assignment #6.

The evidence demonstrates no difference in the programming perseverance promoted by either group programming instructional method in an introductory level programming course. Ho_{10} is regarded as defensible.

Hypothesis Eleven

Ho_{11} : The teaching method employing fixed heterogeneous student programming groups in combination with lectures, when compared to the method employing changing heterogeneous student programming groups in combination with lectures, has no differential effect on student programming efficiency in an introductory computer programming course as measured by:

1. the time spent by the student in designing, coding and debugging the programming assignment (reported by the student, listed as writing time and measured in

hours);

2. the time spent by the student at the computer terminal for the programming assignment (recorded by the computer, listed as terminal time, displayed at logout and measured in hours);

3. the number of runs required by the student for the programming assignment (recorded by the computer, listed as number of runs and displayed as the file generation number).

Students who do not hand in the above data for a program are not included in the analysis for that programming assignment. Table 19 presents the means and standard deviations of the programming efficiency measures for students in Comparison 3.

Table 19

Mean and Standard Deviation of Programming Efficiency Measures Used in Comparison 3

Measure	FHG M	FHG SD	RHG M	RHG SD
Programming Assignment #3				
Writing time	1.5062	.670	2.8444	2.520
Terminal time	1.6437	.614	2.0647	1.521
Number of runs	4.0000	4.775	6.6111	6.204
Programming Assignment #4				
Writing time	1.9000	1.264	2.7000	2.349
Terminal time	2.3937	.920	2.1812	.949
Number of runs	7.1875	7.250	11.0625	7.750
Programming Assignment #5				
Writing time	3.2000	1.461	2.9643	1.599
Terminal time	2.8733	1.242	3.1643	2.023
Number of runs	9.6000	9.448	8.5000	6.959
Programming Assignment #6				
Writing time	2.7000	1.935	2.4333	1.203
Terminal time	3.0200	1.168	2.4250	1.218
Number of runs	7.4000	4.485	9.0625	7.733

Note. Writing time and Terminal time are measured in hours.

The first of three measures used to determine the student programming efficiency is the writing time recorded by the student for each programming assignment. The only writing time to obtain statistically significant difference between the FHG class and the RHG class is the average writing time for the third programming assignment. Analysis of the data using time series displays no recognizable trend.

The second measure of student programming efficiency is the amount of terminal time recorded for each completed programming assignment. Analysis of the data obtained from the students in the FHG and the RHG classes does not find any significant differences.

The third aspect of student programming efficiency used as a measurement is the number of runs required by the student to obtain correct program output. This is recorded by the computer for each assignment. Again analysis of the data from the classes does not obtain any significant differences.

The experimental instruction method employing fixed heterogeneous student programming groups, when compared to the experimental instruction method employing variant heterogeneous student programming groups, does not appear to produce any differential effects as determined byhe three measures of student programming efficiency used in this study. Therefore Ho_{11} is tenable.

Hypothesis Twelve

Ho_{12} : The teaching method employing fixed heterogeneous student programming groups in combination with lectures, when compared to the method of employing changing heterogeneous student programming groups in combination with lectures, has no differential effect on student attitude toward computers and programming as measured by:

1. the Attitude Toward Computers and Programming Survey given pretreatment,

2. the Attitude Toward Computers and Programming Survey given posttreatment (after the sixth programming assignment).

Table 20 presents a descriptive analysis of the average pretreatment and posttreatment item response .

Table 21 and Figure 9 compare the average total student response to the survey, pretreatment and posttreatment, in the FHG and RHG classes.

Table 20

Pretreatment and Posttreatment Mean Item Response on the Attitude Toward Computers and Computer Programming Survey in Comparison 3

	Pretreatment		Posttreatment	
Item	FHG	RHG	FHG	RHG
1	4.4400	4.6842	4.5556●	4.7778●
2	4.4400	4.3158	4.3333	4.3889●
3	4.5200	4.5556	4.3889	4.8889●
4	4.1667	4.3158	4.3333●	4.3529●
5	3.8800	3.9474	4.0000●	4.0000●
6	4.4000	4.6316	4.1667	4.5000
7	3.8000	4.0000	3.5556	3.5882
8	4.0400	4.3684	4.3889●	4.5556●
9	3.8400	4.0516	3.9444●	3.8889
10	3.4800	3.8947	3.6111●	3.7222
11	4.0800	4.1053	4.3333●	4.1667●
12	4.0400	4.2632	4.2778●	4.2222
13	4.2400	4.2105	4.3333●	4.2222●
14	3.7200	4.0526	3.8889●	3.3333
15	4.1200	4.2105	4.2778●	4.3889●
16	3.2800	3.5789	3.2353	3.4444
17	3.7600	3.5789	3.5556	3.7222●
18	3.8800	3.9474	3.9412●	3.5556

Note. A score of 5 indicates a most favorable attitude toward the item.

● Indicates an increase of item mean response posttreatment compared to pretreatment in same class.

Table 21

Mean and Standard Deviation of Pretreatment and Posttreatment Student Total Response Average on the Attitude Toward Computers and Computer Programming Survey in Comparison 3

Survey	FHG		RHG	
Source	M	SD	M	SD
Pretreatment	3.9978	.487	4.1374	.440
Posttreatment	4.0401	.545	4.0154	.459

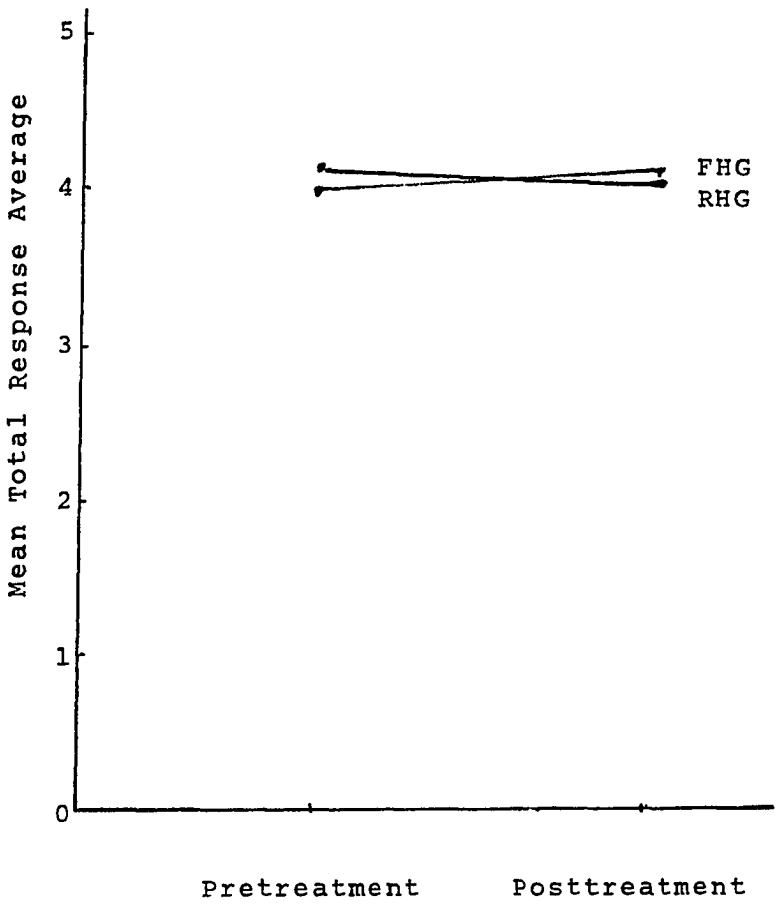


Figure 9. Pretreatment and posttreatment means of student total response average on the Attitude Toward Computers and Computer Programming Survey in the FHG and RHG classes.

Analysis of the responses in the FHG and RHG classes on the pretreament Attitude Toward Computers and Computer Programming Survey finds no statistically significant differences. The RHG class scores slightly more favorable responses to 14 statements on the survey.

Analysis of the responses in the FHG and RHG classes on the posttreatment Attitude Toward Computers and Computer Programming Survey also finds no statistically significant differences. The RHG class scores slightly more favorable responses to 11 statements on the survey.

The RHG class responds more favorably to nine statements on the posttreatment survey compared to the class responses on the pretreatment survey, and the FHG class responds more favorably to 12 statements on the posttreatment survey compared to the class responses on the pretreatment survey. Seven of the statements receiving more favorable class response postreatmentwise are the same for both classes.

Comparison of the total response average for the pretreatment and postreatment surveys reveals no significant differences for the FHG and the RHG classes.

The current investigation fails to detect any significant differences in student attitude in the classes employing either student programming group structure in an introductory programming course and so Ho_{12} is tenable.

The Team Project and Student Reactions to the Course

Programming assignment #7 is a team project in all the classes. It is assigned and completed after the experiment and is used as an aid in the assessment of the data obtained for Comparisons 1, 2 and 3.

Table 22 presents the average student grade received on programming assignment #7 for each class and the percentage of students in each class completing the project.

Table 22

Mean and Standard Deviation of Student Grade Received on Team Project and Percentage of Students Completing the Assignment

Class	M	SD	Percentage
IND	6.0476	3.840	76.2
FHG	8.3500	3.617	85.0
RHG	8.7895	3.119	89.5

Prior to the final examination, the students in the three classes are asked to evaluate the course. One method used requested the students to list the most positive aspect and the most negative aspect of the course. The second method used had the students select one word from each of eight opposite word-pairs to describe the course.

Listed below are some of the student answers to the request to name the most positive and the most negative aspects of the course. The students' responses are presented by class.

The Most Positive Aspect of the Course

IND Class.

"Working in groups."

"Nothing."

"The group project in which we worked together."

"Good experience working with a group. Program (programming assignment #7) . . . looked more difficult than it actually was."

"The project was more interesting because it was shared and more fun."

"I enjoyed working with Dave and Diane. We should have had time to do more programs in groups."

FHG Class.

"Group work."

"I liked working in the group and really learned more that way."

"Walkthroughs with groups are great if everyone has his work done and shows up to help out."

"Everyone in the group was willing to help one another and I find that that is really good in a class like this."

"It was good to get a feeling of working with other people and seeing how they approach problems."

"Last program was interesting with total group involved. Each person participated in order to develop the program and allowed for group discussion in aiming at the final run."

RHG Class.

"Met very helpful and highly motivated people. It made me feel I could do a little bit of programming."

"To see same problem solved in different ways."

"I felt helping others out with their problems with their programs was both challenging and a learning experience."

"Working with others from whom I learned a lot."

"Working with a group to solve a problem."

"Programming Assignment #7 was the most interesting of all . . . fun to work in a group and solve problems collectively."

Most Negative Aspect of the Course

IND Class.

"Getting on the terminals."

"It's a pain! Not enough time! Hard to get people together!"

"Group did not work out . . . too much time waiting for people, trying to meet, etc. One member was unreliable and unresponsive to phone calls."

"Group project because only one other person in my group was willing to work."

"Too much work in too little time."

"Falling behind."

"It actually took me 55 runs to get assignment #5 to work properly."

"It would have been a lot better if the rest of my group knew that this was supposed to be a 'group' project!!"

FHG Class.

"Too much information in too little time."

"The group idea did not work."

"Trying to get the group together outside of class was difficult."

"Not having other group members' work prepared on time. This makes the group situation fairly useless."

RHG Class.

"Course moves very fast; not enough time."

"Being graded as one for group performance."

"The group idea."

"Group programming with a poor group--a very frustrating experience. After a group change, however, it was rewarding."

"Having someone else's program reflect my grade--working with a group that was not cooperative. When all three worked together it was beneficial, but if one did not do the work it was frustrating for the others. Also being a commuter school, it was hard to get together outside of class."

"The frustration of being dependent on others who did not have the same enthusiasm as I had."

"The group idea--I feel that each person should do his or her own work. When grades are concerned, it is unfair to penalize one person for another peron's mistakes. In some instances, group members gave a copy of their program to a member of the group who couldn't do the program. This person copied the others' program and handed it in as his own. If each person is responsible for his/her and only his/her own work, I feel that it would be a much more enjoyable course."

Selecting From Word-Pair Descriptions

Table 23 presents the opposite word-pairs that are offered to the student and the number of students in each class who select each choice.

Table 23

Student Selections From Word-Pair Descriptions for the Course Evaluation

Word-Pair Description	Class IND	FHG	RHG
Easy/hard	4/11	2/12	5/11
Enjoyable/unenjoyable	17/1	12/2	16/1
Requires skill/requires luck	16/0	17/0	16/0
Relaxing/frustrating	3/11	4/9	7/8
Pleasing/unsettling	13/2	9/4	14/2
Learned much/learned little	15/2	16/1	16/1
Simple/challenging	3/15	1/16	1/16
Waste of time/not enough time	0/18	1/15	0/17

Note. Only those students who respond are counted.

The assessment of student reactions to the course by the selection from opposite word-pairs and the listing by the student of the most positive and negative course aspect produces similar results in the IND, FHG and RHG classes. The majority of students in each the three classes find the course to be hard, enjoyable, requiring skill, frustrating (a larger majority in the IND class), pleasing and challenging. Students feel that they learn much, but, that there is not enough time to cover all the material in sufficient detail.

Although the team project is the first and the only exposure the students in the IND class have to a group programming activity, all three classes refect upon the group programmming experience when listing the most positive or the most negative aspect of the course. The group becomes an interesting, rewarding, learning experience when all the members in the group participate, and a frustruating waste of time when the members are uncooperative.

The average student grade on the team project is the highest in the RHG class, in which 89.5% of the students complete the assignment. Again, students who do not complete the assignment are given a grade of zero and are included in the analysis. This may inflat the grade difference observed between the classes. When the results are compared on the basis of only those students who complete the team project, the grade differential

between the classes is reduced.

CHAPTER VI

CONCLUSION

Summary

Computer programming is no longer a mysterious art practiced by a very few. Many countries are rapidly converging to a totally computerized society, making computer education one of the most critical areas of concern. Campuses across the nation are offering, and many are requiring, students to take introductory courses in programming languages. Computer literacy is viewed as a necessary tool for the graduate of today.

Students enter introductory programming courses with a variety of academic backgrounds, levels of maturity, motivations and needs. They may be contemplating a computer science major, fulfilling a requirement for another major, or seeking self-enlightenment. The problem of how to teach to such a wide audience of students has become a most important concern of computer science education.

The purpose of this study is to investigate the effects of instructional methods employing group programming on student programming skills and student attitude in an introductory level computer programming

course. Although computer science instructors are employing many innovative methods in introductory computer programming courses, none of the methods appear to document substanially the consistent improvement of these students cognitively or affectively. Consequently the traditional pedagogical method, combining lecture with individual programming, continues to have widespread use.

Claiming that the traditional instructional method practiced in introductory level courses, fosters frustration and discouragement, computer psychologists advocate the use of student programming groups. Students in the group work together to analyze and design solutions to programming assignments and the group setting offers student members the opportunity to see other coded solutions and have theirs reviewed by group members for error detection.

It is expected that these activities reduce the time requirement and frustration element associated with introductory level programming courses that result in a high percentage of the students not completing the course; and, increase the programming skill and promote positive attitudes toward computers for the student.

Socio-psychologists have found that a well-functioning, cohesive group has major implications for cognitive stimulation and attitude influence when the individual wants to belong to the group and perceives

self as a member of the group. Furthermore, attitude changes, resulting from group membership, have been linked to changes in behavior and it is claimed that the development of a positive attitude toward a subject can result in increased cognitive achievement and in pursuance of further study in the subject.

Research on all levels of education find the group to be a powerful tool in the problem solving situation. In cooperative learning, students share information, generate alternate ideas and sharpen their inference through discussion. Besides increasing confidence and self-esteem by eliminating the detrimental effects of competition (i.e., anxiety, indifference, resentment), groups have been found to offer support to the student, often resulting in increased learning and a more positive attitude toward the subject.

This study is designed to examine two experimental instructional methods that combine lectures with student programming groups. One method combines lectures with fixed heterogeneous student programming groups and the other combines lectures with variant heterogeneous student programming groups. The investigation is organized according to three comparisons:

Comparison 1 examines the programming skills, defined as programming proficiency, programming perseverance and programming efficiency, and student attitudes in a class receiving instruction in the

traditional method as they compare to those of students in a class receiving instruction in the experimental method employing fixed heterogeneous programming groups.

Comparison 2 examines the programming skills and attitudes of students in a class receiving instruction in the traditional method compared to those of students in a class receiving instruction in the experimental method employing variant heterogeneous programming groups.

Comparison 3 examines the programming skills and attitudes of the students in the classes receiving instruction in the two experimental modes.

The study is conducted at a private four-year, coeducational college that services approximately 10,000 students under a policy of open enrollment. Three daytime sections of an introductory level computer programming course entitled Structured Programming Using FORTRAN are selected for the investigation.

The traditional instructional mode (IND) employing individual programming and the experimental instructional mode employing fixed student programming groups (FHG) are randomly assigned to the two weekday sections; the experimental instructional mode employing variant student programming groups (RHG) is assigned to the third section that meets on Saturday.

The comparability of the students in the classes is assessed on 18 items dealing with the sex, age, educational background, computer background, general

ability, programming ability, educational commitment and working commitment. The IND and the FHG classes differ significantly on the number of credits being taken during the semester. This indicates that there are more full-time students in the IND class than the FHG class. The IND and the RHG classes differed significantly on three items in the assessment: number of credits being taken during the semester, age and workload. The IND class appears to have a strong commitment to school responsibilities and the RHG class appears to have a strong commitment to work responsibilities outside of school. (These differences are anticipated since the RHG class meets on the weekend; and so the main comparative study is assigned to the two weekday classes.) The FHG and the RHG classes differ signifcantly on the work commitment only.

Students in the FHG and RHG classes are placed into three classifications (i.e., good, average, weak) of programming skill based upon the score they earn on the Preliminary Programming Proficiency Test and their previous programming experience. One student from each classification is selected for each heterogeneous programming group. These groups remain fixed in the FHG class and change after every programming assignment in the RHG class. Some groups reduce to two members or combine to become four-member groups when students are absent or leave the course.

The group programming process for the FHG and RHG classes consists of three in-class sessions on each programming assignment. The first session is for the analysis of the problem and the design of an algorithmic solution. The second session is for critical review of the first run of the coded version of the solution that is written by each group member. The third session is a discussion and review of the final run of each member's program for that assignment. The time spent in group sessions in the experimental classes is spent by the control class in designing, analyzing and desk-checking (see Footnote 4) the programming assignments individually.

All students in the study are requested to hand in a summary sheet with each programming assignment they complete. The summary is to include the time spent by the student in designing, coding and debugging the program, the time spent on the terminal, and the number of runs required by the program to produce correct output. The experiment begins with the third programming assignment and ends with the sixth programming assignment.

The first hypothesis in Comparisons 1, 2 and 3 examines student programming proficiency as measured by the grades received on the programming assignments and the scores earned on the midterm and final examinations.

The second hypothesis in Comparisons 1, 2 and 3

examines student programming perseverance as measured by the number of students completing each assignment on time, the number of students completing each assignment late, and the number of students not completing each programming assignment.

The third hypothesis in Comparisons 1, 2 and 3 examines student programming efficiency as measured by the time spent by the student in designing, writing and debugging the program, the time spent by the student on the terminal and the number of runs required to obtain a correct output for each programming assignment.

The fourth hypothesis in Comparisons 1, 2 and 3 examines the student attitude toward computers and programming as measured by an Attitude Toward Computers and Programming Survey taken pretreatment and posttreatment.

To aid in the assessment of the data obtained in the hypotheses listed above, student reactions to the course are solicitated and a seventh programming assignment, a team project, is required of all students. In the IND class, students are placed in heterogeneous teams based on the grades they have earned in the course at that time. The teams in the other two classes are selected as usual.

Results

The following results are presented in response to the four questions proposed for investigation in the introductory chapter of the study.

1. Do students who receive instruction in the fixed group (FHG) or variant group (RHG) or traditional setting (IND) achieve a higher level of programming proficiency?

The students in the FHG and RHG classes perform better on the programming assignments than do the students in the IND class. However, the differences on the midterm and final examinations, on which the experimental classes score lower than or similarly to the control class, are not found to be significant.

A plausible explanation for this disparity may lie in the fact that the programming group carries its weakest member through the programming assignments in order to raise the proficiency level of the group as a whole. (Students programming in groups are operating under the premise that their grades reflect the average of the group grades.) However this support from the group does not exist for the student during the midterm or final examination which is taken individually.

Consideration must also be given to the support which the weak student receives from the group that may cause him/her to remain in the course for a longer period of time than would be the case had this student been in a

class that is not employing groups.

Hypotheses 1, 5 and 9 suggest that the level of programming proficiency is comparable for students in the three classes in the study.

2. Does the individual operating within the fixed programming group or the variant programming group or programming alone exhibit greater programming perseverance?

The students in the FHG and RHG classes complete more programming assignments on time than do the students in the IND class. The programming perseverance measurement is considered an important factor in the evaluation of a course that has as one of its primary objectives the production of computer-literate students. If a student is not completing the assignments, that student is not keeping pace with the work in the course and is, for all practical purposes, not continuing with the course.

The observed trend, in the IND class, of fewer students completing each programming assignment as the course progresses is reversed for the seventh programming assignment, the team project. This may be considered further evidence for the programming perseverance promoted by programming within the group environment in an introductory level course.

Hypotheses 2 and 6 determine the completion differences between the experimental classes and the

control class to be significant and find the students programming within the group to exhibit greater programming perseverance than students programming alone.

The students in each of the experimental classes complete a similar number of programming assignments and Hypothesis 10 suggests that the programming perseverance of these students is comparable.

3. Do students who work within the fixed group structure or the variant group structure or alone make more efficient use of their time (pertinant to the course) and computer facilities?

Students in the FHG and RHG classes spend more time writing programs and using terminals then do students in the IND class. The number of runs required per assignment for most students in the experimental classes is less than that required by the students in the control class.

The greater amount of time spent writing and using the terminals by the students in the experimental classes appears to be the results of a higher percentage of the students in the FHG and RHG classes reporting this data, rather than any increase or decrease in programming efficiency gained from the group programming experience. The fewer runs per programming assignment that are required by the students in the experimental classes may be considered a result of having students review group members' programs before the programs are presented to

the computer.

However Hypotheses 3, 7 and 11 are unable to substaniate any of these differences as significant for all the programming assignments and the use of student time and computer facilities by the students in the three classes is considered to be comparable.

4. Do students who program within the fixed group or variant group or traditional environment exhibit a more positive attitude toward computers and computer programming?

In the posttreatment survey, the students in the FHG and RHG classes demonstrate significantly more positive attitudes toward computers and computer programming than do students in the IND class. The total response averages of the experimental classes are similar.

Hypotheses 4 and 8 determine that students programming within the group environment exhibit a more positive attitude toward computers and computer programming than students who program alone. Hypothesis 12 suggests that the attitudes of students programming within variant group and fixed group environments are comparable.

In summary, the investigation finds no significant differential effects are produced by any of the methods on measures of programming proficiency or programming efficiency. However, in introductory courses, in which computer literacy is a main course objective, the more

positive effects substantiated for measures of programming perseverance and student attitude in the classes employing student programming groups, lead to the conclusion that these instructional methods are worthy of consideration for courses designed for student populations as described in this study.

It must be noted, however, that the results of Comparisons 1, 2 and 3 are derived from the responses of a limited representation of the subject population and to a subjective set of measures of student programming skills and attitude. Consequently, although the results are promising, they may hardly be considered conclusive.

Recommendations

Based on the data obtained under the conditions of this study, the following suggestions for further research are made:

1. Replication is of prime importance since any similar findings would provide further support for the conclusions of this study. The replication studies should be conducted using:

a. Similar experimental format with different samples of students from the same population;

b. Similar experimental format with different student populations: traditional college students, dormitory students, graduate students, computer science

majors, non-computer science majors;

c. Similar experimental format with different programming lanuages: PL/I, BASIC, PASCAL; and,

d. Similar experimental format with different methods of group formation and different group size: self-selection, 2-member groups, 5-member groups.

2. The development or identification of accurate methods to measure student programming proficiency would aid in the evaluation of differences resulting when instructional methods are compared. These may include:

a. Requirement of more or fewer written programming assignments;

b. Implementation of more individually taken examinations;

c. Introduction of oral quizzes on each programming assignment;

d. Assignment of programs to be modified or debugged;and,

e. Employment of measures of program readability as a determinant of programming skill.

3. A search for unobtrusive and accurate measures of student programming efficiency would aid in the evaluation of differences resulting from a comparison of instructional methods. These may include:

a. Examination and cataloguing of student programming errors; and,

b. Determination of ability of the group as

compared to the individual to detect different types of errors.

4. General research into areas not considered by this study would add to the knowledge of the effects of student programming groups. This may include:

a. A study of group dynamics as it affects the group process in student programming groups (i.e., personality factors and group member roles, different personality combinations in student programming groups);

b. The employment of a clinical-type research design in which the investigator becomes a programming group participant in order to observe the positive and negative features of social interaction in group problem solving;

c. The investigation of different instructional methods as they relate to different cognitive learning styles; and,

d. The identification of students who prefer to work alone and the investigation of the detrimental effects caused by those students on the functioning of a programming group in a class employing the changing group structure.

It is hoped that the observed benefits obtained in the experimental classes employing student programming groups will give encouragement to future teaching experiments conducted in introductory level computer programming classrooms. Studies of pedagogical methods

which maximize positive student attitude and perseverance should be as serious a matter of concern to the investigator as those methods that maximize conveyance of knowledge. This, together with the continued publication of such research, offers promise of finding the best pedagogical method for producing computer literate students.

References

Ahlgren, D., Sapega, A., & Warner, H. A. Sequence of computing courses for liberal arts colleges. SIGCSE Bulletin, 1978, 10(1), 180-182.

Ahmed, N., & Bardos A. Programmers' mass education at Szamok. SIGCSE Bulletin, 1976, 8(2), 43-47.

Aiken, J. A self-paced first course in computer science. SIGCSE Bulletin, 1981, 13(1), 78-85.

Aleck, A. Guidance: A new dimension of creative teaching. In C. Skinner (Ed.), Educational psychology. Englewood Cliffs: Prentice-Hall, 1959.

Alford, M. Hsia, P., & Petry, F. A software engineering approach to introductory programming courses. SIGCSE Bulletin, 1977, 9(1), 157-161.

Allport, G. Attitudes. In C. Murchison (Ed.), A handbook of social psychology. Worchester, Mass.: Clark University Press, 1935.

Arnow, B. Realism in the classroom--a team approach. SIGCSE Bulletin, 1981, 13(2), 5-11.

Aronson, E. The jigsaw classroom. Beverly Hills, Calif.: Sage, 1978.

Artzt, A. Student teams in mathematics class. The Mathematics Teacher, 1979, 72, 505-508.

Bain, R. An attitude on attitude research. American Journal of Sociology, 1928, 33, 940-957.

Baker, F. T. System quality through structured programming. AFIPS Proceedings, 1970, 339-343.

Baker, F. T. Chief programmer team management of production programming. IBM Systems Journal, 1971, 1, 56-73.

Barnett, M. Systematic Instruction in Simple Programming Gambits. SIGCSE Bulletin, 1978, 10(3), 108-112.

Basili, V. R., & Reiter, R. W., Jr. A controlled experiment quantitatively comparing software development approaches. IEEE Transactions on Software Engineering, 1981, 7(3), 299-320.

Bell, F. Can computers really improve school mathematics? The Mathematics Teacher, 1978a, 71, 428-433.

Bell, F. H. Teaching and learning mathematics (In secondary schools). Dubuque, Iowa: Wm. C. Brown, 1978b.

Bentler, P. M. & Speckart, G. Models of attitude-behavior relations. Psychological Review, 1979, 86, 452-464.

Bezanson, W. R. Teaching structured programming in FORTRAN with IFTRAN. SIGCSE Bulletin, 1975, 7(1), 196-199.

Bohl, M. Information Processing with BASIC (3rd ed.). Chicago: Science Research Associates, 1982.

Brooks, F. P. Jr. The mythical man-month: Essays on software engineering. Reading, Mass.: Addison-Wesley, 1975.

Brown, G. I. The live classroom. New York: Viking, 1975.

Bruce, W. F. Personality and children's adjustment problems. In C. E. Skinner (Ed.), Educational psychology (4th ed.). Englewood: Prentice-Hall, 1959.

Bruner, J. S. Toward a theory of instruction. Cambridge: Belknap Press of Harvard University, 1966.

Buck, J., & Shneiderman, B. An internship in information systems: Combining computer science education with realistic problems. SIGCSE Bulletin, 1976, 8(3), 80-83.

Cashman, W. F., & Mein, W. J. On need for teaching problem-solving in a computer science curriculum. SIGCSE Bulletin, 1975, 7(1), 40-46.

Cheng, R. On-line large screen display system for computer instruction. SIGCSE Bulletin, 1976, 8(1), 366-370.

Chi, E. C., Morah, M., & Tausner, M. R. Computer science at Staten Island Community College. SIGCSE Bulletin, 1974, 6(1), 48-52.

Comaa, H., Kramer, J., & Penney, B. K. A student group project in operating system implementation. SIGCSE Bulletin, 1978, 10(1), 197-202.

Conway, R. W. Introductory instruction in programming. SIGCSE Bulletin, 1974, 6(1), 6-10.

Cook, C. R. A self-paced introductory FORTRAN programming course. SIGCSE Bulletin, 1976, 8(3), 78-79.

Cook, C. R. Applications programming course using guided design. SIGCSE Bulletin, 1977, 9(3), 79-82.

Cooper, R. T., & Lane, M. G. An improved hands-on approach to teaching systems programming and the impact of structured programming. SIGCSE Bulletin, 1976, 8(3), 115-124.

Costello, D. F., & Schonberger, R. J. On guiding the business school toward computer literacy. SIGCSE Bulletin, 1977, 9(1), 180-183.

Crenshaw, J. H. Team projects in the undergraduate curriculum. SIGCSE Bulletin, 1978, 10(1), 203-205.

Daly, C., Embley, D. W., & Nagy, G. A progress report on teaching programming to business students without lectures. SIGCSE Bulletin, 1979, 11(1), 247-250.

Danielson, R. L., & Nievergelt, J. An automatic tutor for introductory programming students. SIGCSE Bulletin, 1975, 7(1), 47-50.

Davidson, D. Learning mathematics in a group situation. The Mathematics Teacher, 1974, 67, 101-106.

Davidson, N., Agreen, L., & Davis, C. Small group learning in junior high school mathematics. School Science and Mathematics, 1978, 78, 23-30.

Deimal, L. E. Jr., & Prozefsky, M. Requirements for student programs in the undergraduate computer science curriculum: How much is enough. SIGCSE Bulletin, 1979, 11(1), 11-17.

Dersham, H. L. A modular introductory computer science course. SIGCSE Bulletin, 1981, 13(1), 177-181.

Dinerstein, N. T. Does computer science belong in a liberal arts college. SIGCSE Bulletin, 1975, 7(2), 55-64.

Eccles, W. J., & Gordon, B. G. Computer science by TV. SIGCSE Bulletin, 1976, 8(3), 54-56.

Eckberg, C. F. Some proposals for distributing central computing power at a university. SIGCSE Bulletin, 1976, 8(3), 129-134.

Ettinger, H. A., Goodman, G. I., & Plumm, C. FORTRAN: A self-paced, mastery-based course. SIGCSE Bulletin, 1981, 13(1), 62-73.

Exline, R. V. Explorations in the process of person perception: Visual interaction in relation to competition, sex, and need for affiliation. Journal of Personality, 1963, 31, 1-20.

Farley, P., Grossman, E., & Tucciarone, J. Mastering BASIC--a beginner's guide. Elmsford, New York: Collegium, 1979.

Fishbein, M., & Ajzen, I. Attitudes toward objects as predictors of single and multiple behavioral criteria. Psychological Review, 1974, 81, 59-74.

Fishbein, M., & Ajzen, I. Beliefs, attitude, intention and behavior: An introduction to theory and research. Reading, Mass.: Addison-Wesley, 1975.

Fiske, E. B. Schools re-evaluate arithmetic. New York Times, August 12, 1980.

Fletcher, F. M. An approach to the problem of teaching effectiveness. In W. J. McKeachie (Ed.), The appraisal of teaching in large universities. Ann Arbor: University of Michigan, 1958.

Fosdick, H., & Mackey, K. A course on the Pragmatic tools of the programming environment: Description and rationale. SIGCSE Bulletin, 1979, 11(3), 11-14.

Freeman, P. Realism, style and design: Packing it into a constrained course. SIGCSE Bulletin, 1976, 8(2), 150-157.

Friedman, F. L., & Koffman, E. B. Teaching problem solving and structured programming in FORTRAN. SIGCSE Bulletin, 1977, 9(1), 63-68.

Garibaldi, A. Affective contributions of cooperative and group goal structures. Journal of Educational Psychology, 1979, 71, 788-794.

Gates, A. I. Educational psychology (3rd ed.). New York: Macmillan, 1948.

Geffner, R. The effects of interdependent learning on self-esteem, interethnic relations, and intra-ethnic attitudes of elementary school children. A field experiment. Unpublished doctoral dissertation, University of California, Santa Cruz, 1978.

Gillett, W. An interactive program advising system. SIGCSE Bulletin, 1976, 8(1), 335-341.

Goldman, M. A comparison of individual and group performance for varying combinations of initial ability. Journal of Personality and Social Psychology, 1965, 3(1), 210-216.

Goodman, D. A., & Crouch, J. Effects of competition on learning, ICUT, 1978, 26(2), 130-133.

Grier, S. A tool that detects plagarism in PASCAL programs. SIGCSE Bulletin, 1981, 13(1), 15-20.

Gries, D. What should we teach in an introductory progamming course. SIGCSE Bulletin, 1974, 6(1), 81-89.

Grossman, E., & Tucciarone, J. Mastering micro BASIC. Pleasantville, New York: Comtext, 1981.

Guha, R. K., Carr, P. A., & Smith, C. L. Standards considered helpful. SIGCSE Bulletin, 1977, 9(3), 73-78.

Gunderson, B., & Johnson, D. Building positive attitudes by using cooperative learning groups. Foreign Language Annals, 1980, 13(1), 39-43.

Gurnee, H. A comparison of collective and individual judgments of facts. Journal of Experimental Psychology, 1937, 21, 106-11.

Hartley, E. L. Attitude research and the jangle fallacy. In C. Sherif & M. Sherif (Eds.), Attitude, ego-involvement, and change. New York: John Wiley and Sons, 1968.

Hoffman, L. R. Applying experimental research on group problem solving to organization. The Journal of Applied Behavioral Science, 1979, 15, 375-391.

Homeyer, F. C. An experimental microcomputer course (A case history). SIGCSE Bulletin, 1977, 9(4), 41-44.

Horowitz, E. L., & Horowitz, R. E. Development of social attitudes in children. Sociometry, 1938, 1, 301-338.

Hovland, C. I., Janis, I. L., & Kelley, H. H. Communication and persuasion. New Haven, Conn.: Yale University Press, 1953.

Irby, T. C. Teaching software development using a microprocessor laboratory. SIGCSE Bulletin, 1977, 9(1), 113-118.

Jehn, L. A., Rine, D. C., & Sondak, N. Computer science and engineering education: Current trends, new dimensions and related professional programs. SIGCSE Bulletin, 1978, 10(3), 162-168.

Jewell, L. N., & Reitz, J. Group effectiveness in organizations. Glenview, Ill.: Scott, Foresman, 1981.

Johnson, D. C., Anderson, R. E, Hansen, T. P., & Klassen, D. L. Computer literacy--what is it The Mathematics Teacher, 1980, 73, 91-96.

Johnson, D. W., & Johnson, R. T. Learning together and alone, cooperation, competition and individualization. Englewood Cliffs, New Jersey: Prentice Hall, 1975.

Johnson, D. W., & Johnson, R. T. Cooperative, competitive and individualistic interdependence in the classroom. Journal of Reasearch and Development in Education, 1978, 12, 3-15.

Johnson, D. W., Johnson, R. T., Johnson, J., & Anderson. Effects of cooperative versus individualized instruction on student prosocial behavior, attitudes toward learning and achievement. Journal of Educational Psychology, 1976, 68, 446-452.

Johnson, R. T., & Johnson, D. W. Cooperative learning, powerful sciencing. Science and Children, 1979, Nov-Dec, 26-27.

Kelley, H., & Thibaut, J. W. Experimental studies of group problem solving and process. In G. Lindzey (Ed.), Handbook on social psychology (Vol. 2). Cambridge, Mass.: Addison-Wesley, 1954.

Kelman, H. C. Compliance, identification and internalization: Three processes of attitude change. Journal of Conflict Resolution, 1958, 2, 51-60.

Kenworthy, D. J., & Redish, K. A. Software team projects. SIGCSE Bulletin, 1979, 11(1), 37-40.

Kernighan, B. W., & Plauger, P. J. Programming style. SIGCSE Bulletin, 1974, 6(1), 90-96.

Khailany, A. Advanced structured COBOL programming. SIGCSE Bulletin, 1977a, 9(1), 59-62.

Khailany, A. Alternative teaching strategy for an introductory computer language course. SIGCSE Bulletin, 1977b, 9(1), 93-95.

Khailany, A., & Holland, R. H. An introductory computer course in a school of business. SIGCSE Bulletin, 1975, 7(2), 39-42.

Khailany, A., & Saxon, C. Conducting project team classes in data processing. SIGCSE Bulletin, 1978, 10(1), 189-192.

Kiesler, C. A., Collins, B. E., & Miller, N. Attitude change--a critical analysis of theoretical approaches. New York: John Wiley and Sons, 1969.

Kimura, T. Reading before composition. SIGCSE Bulletin, 1979, 11(1), 162-166.

King, V. G. A confluent approach to nursing education through group process. Nurse Educator, 1978, 5(3), 20-25.

Korfhage, R. A., & Smith, R. J. Individualized instruction in computer science. SIGCSE Bulletin, 1974, 6(1), 161-164.

Lamm, H., & Trommsdorff, G. Group versus individual performance on tasks requiring ideational proficiency (brainstorming): A review. European Journal of Social Psychology, 1973, 3, 361-388.

Lane, M. G. A hands-on approach to teaching systems programming. SIGCSE Bulletin, 1975, 7(1), 23-30.

Lemos, R. S. A comparative study of the effectiveness of team interaction in COBOL programming language learning. Unpublished doctoral dissertation, UCLA, 1977.

Lemos, R. S. The cost-effectiveness of team debugging in teaching COBOL programming. SIGCSE Bulletin, 1978, 10(1), 193-196.

Lemos, R. S. An implementation of structured walk-throughs in teaching COBOL programming. Communications of the ACM, 1979a, 22(6), 335-340.

Lemos, R. S. Teaching programming languages: A survey of approaches. SIGCSE Bulletin, 1979b, 11(1), 174-181.

Lightner, S. M. Accounting education and participatory group dynamics. Collegiate News and Views, 1981, 35(1), 5-9.

Likert, R. A. A technique for the measurement of attitudes. Archives of Psychology, 1932, No. 140, 1-55.

Linder, W. Computer-tutor: From a student project to a self-paced CAI/CMI course. SIGCSE Bulletin, 1976, 8(3), 57-60.

Little, J. ACM committee on curriculum in computer science recommendations for the undergraduate program in computer science. SIGCSE Bulletin, 1977, 9(2), 1-16.

Lucas, W. R. Planned attitude change while teaching computer literacy. SIGCSE Bulletin, 1976, 8(1), 90-94.

Mackey, K., & Fosdick, H. An applied computer science/systems programming approach to teaching data structures. SIGCSE Bulletin, 1979, 11(1), 76-78.

Maier, N. R. F, & Solem, A. R. The contributions of a discussion leader to the quality of group thinking: The effective use of minority opinions. Human Relations, 1952, 5, 277-288.

Martin, D. C., & Doucouis, A. J. An evaluation of the in-service computer mathematics program, 1960-1962. Project on Information Processing Newsletter, 1964, 2(2), 11-15.

Mathis, R. F. Teaching debugging. SIGCSE Bulletin, 1974, 6(1), 59-63.

Mavaddat, F. An experiment in teaching programming languages. SIGCSE Bulletin, 1976, 8(2), 45-55.

McKuen R., & Davidson, N. An alternative to individual instruction in mathematics. The American Mathematical Monthly, 1975, 82, 1006-1009.

McLeod, D. B., & Adams, V. M. The interaction of field independence with small-group instruction mathematics. Journal of Experimental Education, Winter 79/80, 48(2), 118-124.

Menninga, L. D. Introduction of practical experience into curriculum '68 through integration of courses. SIGCSE Bulletin, 1974, 6(1), 152-154.

Miller, N. E., & Petersen, C. G. An evaluation scheme for a comparison of computer science curricula with ACM's guidelines. SIGCSE Bulletin, 1981, 13(1), 216-223.

Mize, J. L. Making an academic curriculum relevant to business requirements. SIGCSE Bulletin, 1976, 8(2), 24-27.

Moccido, M. R. Teacher training in computer science education in western Australia: Group projects. SIGCSE Bulletin, 1978, 10(1), 206-209.

Newman, J. R. Alternative teaching techniques in computer science. SIGCSE Bulletin, 1973, 5(4), 29-32.

Nievergelt, J. Appropriate areas of computer science research in problems in education. SIGCSE Bulletin, 1974, 6(1), 46.

Noonan, R. E. The second course in computer programming: Some principles and consequences. SIGCSE Bulletin, 1979, 11(1), 187-191.

Ogdin, J. L. The mongolian hordes versus superprogrammer. Infosystems, 12/72, 20-23.

Overall J. U., & Marsh, H. W. Midterm feedback from students: Its relationship to instructional improvement and students' cognitive and affective outcomes. Educational Psychology, 1980, 71, 856-863.

Perry, J. M., & Sondak, N. E. The project experiment in undergraduate computer science education. SIGCSE Bulletin, 1978, 10(2), 21-30.

Perry, J. T., & Weymouth, T. E. A modified CP team approach to an operating system class project. SIGCSE Bulletin, 1975, 7(1), 31-39.

Philipsen, G., Mulac, A., & Dietrich, D. Effects of social interaction on group idea generation. Communications Monographs, 1979, 46, 119-125.

Piaget, J. Science of education and the psychology of the child. New York: Viking Press, 1971.

Piaget, J. To understand is to invent: The future of education. New York: Grossman Publishers, 1973.

Plum, T. W.-S., & Weinberg, G. M. Teaching structured programming attitudes, even in APL, by example. SIGCSE Bulletin, 1974, 6(1), 133-143.

Porter, C. H., & Nesa, L. W. Programming for terminal applications. SIGCSE Bulletin, 1975, 7(3), 77-82.

Prather, R. E., & Schlesinger, J. D. A lecture/laboratory approach to the first course in programming. SIGCSE Bulletin, 1978, 10(1), 115-124.

Ripley, G. D. A course in effective programming. SIGCSE Bulletin, 1975, 7(1), 102-108.

Rosenberg, I. Introductory computer science courses: A modular design. SIGCSE Bulletin, 1976, 8(1), 62-64.

Roth, R. W. Students and faculty training in systems analysis. SIGCSE Bulletin, 1975, 7(2), 67-70.

Ruschitzka, M. An operating systems implementation project for an undergraduate course. SIGCSE Bulletin, 1977, 9(1), 77-84.

Sackman, H. Man-computer problem solving: Experimental evaluation of time-sharing and batch processing. Princeton, New Jersey: Auerbach, 1970.

Sanders, D. H. Computers in business (4th ed.). New York: McGraw-Hill, 1979.

Schmuck, R. A., & Schmuck, P. A. Group processes in the classroom. Dubuque, Iowa: William C. Brown, 1971.

Schribner, T. J. Industry reaction to computer science education. SIGCSE Bulletin, 1974, 6(1), 78-79.

Schulman, E. L. Turning on the undergraduate computer science student: A Re-IPL suggestion. SIGCSE Bulletin, 1977, 9(1), 178-179.

Scott, W. A. Attitude measurement. In G. Linzey & E. Aronson (Eds.), Handbook of social psychology (Vol. 2). Cambridge, Mass.: Addison-Wesley, 1968.

Sebaugh, J. L. The stepwise approach to introductory programming projects with examples. SIGCSE Bulletin, 1976, 8(1), 372-381.

Senn, J. A. A problem oriented pedagogy for computer language. SIGCSE Bulletin, 1974, 6(4), 26-29.

Senn, J. A., & Ives, B. Behavioral education requisites for application-oriented computer scientists. SIGCSE Bulletin, 1979, 11(1), 195-201.

Sharan, S. Cooperative learning in small groups: Recent methods and effects on achievement, attitudes, and ethnic relations. Review of Educational Research, 1980, 50(2), 241-271.

Shaw, M. E., & Wright, J. M. Scales for measurement of attitudes. New York: McGraw-Hill, 1967.

Shelly, G. B., & Cashman, T. J. Implementation of structured walkthroughs in the classroom. The Compiler, Fall 1977, 11-15.

Sherif, C. W, & Sherif, M. Attitude, ego-involvement, and change. New York: John Wiley and Sons, 1968.

Shneiderman, B. Group processes in programming. Datamation, 1980, 26(1), 138-141.

Singhania, R. P. Issues in teaching the introductory course in computers in business curriculem. AEDS Journal, Fall 1980, 45-51.

Sjoerdsma, T. An interactive pseudo-assembler for introductory computer science. SIGCSE Bulletin, 1976, 8(1), 342-349.

Slavin, R. Effects of student teams and peer tutoring on academic achievement and time-on-task (Report No. 253, Center for Social Organization). Baltimore, Maryland: The John Hopkins University, 1978.

Spence, J. W., & Groin, J. C. Systems analysis and design--a computer science curriculum. SIGCSE Bulletin, 1978, 10(4), 24-27.

Stanford, G., & Roark, A. E. Human interaction in education. Boston: Allyn and Bacon, 1974.

Strang, R. Group activities in college and secondary school. New York: Harper and Brothers, 1941.

Tam, W. C., & Busenberg, S. N. Practical experience in top-down structured software production in an academic setting. SIGCSE Bulletin, 1977, 9(1), 32-36.

Teague, D. B. Computer programming II, a project-oriented course. SIGCSE Bulletin, 1981, 13(1), 41-45.

Thiabut, J. W., & Kelley, H. H. The social psychology of groups. New York: Wiley and Sons, 1959.

Thomas, J. C., & Carroll, J. M. The psychological study of design (Research Report No. RC 7695, #33220). Yorktown Heights, New York: IBM Research Division, June 1, 1979.

Tsai, S. W., & Pohl, N. F. Student achievement in computer programming: Lecture vs. computer-aided instruction. Journal of Experimental Education, 1977, 46(2), 66-70.

Unger. E. A., & Ahmed, N. An instructionally acceptable cost effective approach to a general introductory course. SIGCSE Bulletin, 1976, 8(2), 28-31.

Wainwright, R. A survey of faculty computer experience, usage, needs, literacy and attitude. SIGCSE Bulletin, 1979, 11(2), 27-35.

Weaver, A. C. Microcomputers in the computer science curriculum. SIGCSE Bulletin, 1978. 10(1), 171-176.

Weinberg, C. Price of competition. Teachers College Record, 1966, 67, 106-114.

Weinberg, G. M. The psychology of computer programming. New York: Van Nostrand Reinhold, 1971.

Williams, K. An experimental course in advanced programming methods. SIGCSE Bulletin, 1976, 8(4), 15-18.

Zajonc, R. B. Social facilitation. Science, 1962, 149, 269-274.

Zimbardo, P., & Ebbesen, B. Influencing attitudes and changing behavior. Reading, Mass.: Addison-Wesley, 1970.

Znaniecki, F. Social groups as products of participating individuals. American Journal of Sociology, 1939, 44, 799-811.

Reference Notes

1. University of Nebraska Computer Network Newsletter. Lincoln, Nebraska, March, 1979.

2. Goldberg, D. Writing proofs cooperatively in small groups: An exploratory study. Paper presented at the NCTM meeting, Cherry Hill, New Jersey, Spring 1980.

3. DeVries, D., & Edward, K. Expectancy theory and cooperation--competition in the classroom. Paper presented at the annual convention of the American Psychological Association, New orleans, 1974.

4. Sharan, S., & Lazarowitz, R. Effects of an unstructured change program on teachers' behavior, attitudes and perceptions. Manuscript submitted for publication, 1980.

5. Born, D. G. Instructors manual for development of a personalized instruction course. Salt Lake City: University of Utah, 1970.

6. Weinberg, G. M. Maintaining success through formal technical reviews. Ethnotech Inc., P.O. Box 6627, 4333 So. 48th, Lincoln, Nebraska 68506.

7. Hazen, M. Evaluation of an experiment in introductory programming instruction. Lincoln: Department of Computer Science, University of Nebraska, May 1979.

8. Suinn, R. M. Mathematical Anxiety Rating Survey. Fort Collins, Colorado: RMBSI, Inc., P.O. Box 1066, 1972.

9. Minnesota Computer Literacy and Awareness Assessment. Minnesota Educational Computing Consortium, 1979.

10. Bullock, D. E. Survey of entering students at Mercy College, Office For Planning and Institutional Research, Mercy College, Dobbs Ferry, New York, April 1981.

Appendix A: Survey of Entering Students--Fall 1981 (Bullock, Note 10)

TABLE 24

Age Distribution of Respondents (%)
Fall 1981

Age Cohort	Men	Women	Total
18 years	12.7%	10.1%	11.2%
18 - 22 years	57.5	54.9	55.8
23 - 25 years	10.9	8.1	9.4
26 - 30 years	9.1	8.9	9.2
31 - 40 years	8.1	10.8	9.7
41 - 50 years	1.5	6.0	3.9
51 - 60 years	0.3	0.8	0.6
61 and over	0.0	0.4	0.2
TOTAL	100.0	100.0	100.0%

A review of the data indicates that more than two out of five respondents are non-white. This fact holds across all variables including gender, age and enrollment status. The following table shows the racial/ethnic background of the respondents.

TABLE 25

Racial/Ethnic Background
Fall 1981

Classification	Percentage
American Indian	0.8
Asian/Pacific Islander	1.0
Black	17.6
Hispanic	23.0
White	53.8
Other	3.9

Fifty-three people indicated that they have some form of permanent handicap. Approximately forty percent (39.6) of these students listed "restricted vision" as their impairment. Seventy-two percent of this group are enrolled on a full-time basis.

The respondents identified various teaching sites as their "home campus" in the following proportions:

Dobbs Ferry	57.9%
Yorktown Heights	4.6%
White Plains	2.2%
Bronx	17.3%
Yonkers	15.7%
Peekskill	2.3%

TABLE 26

Entering Class Most Important Goals

Goals	Primary	Secondary	Tertiary	Total	Weighted Rank
Obtain certificate/degree	44.6	21.9	5.2	71.7	1
Increase academic knowledge	29.1	19.0	5.5	53.6	2
Improve technical skills	4.0	14.2	11.3	29.5	3
Complete courses for transfer	2.4	3.8	16.4	22.6	4
Enrich daily life	3.6	5.6	12.2	21.4	5
Ability to be independent	3.2	5.5	11.6	20.3	6
Formulate long-term career plans	3.1	7.5	6.5	17.1	7
Discover career interest	4.0	4.8	3.6	12.4	8
Prepare for new career	2.0	6.3	3.7	12.0	9
Increase self-confidence	1.8	2.7	3.6	8.1	10

More detailed analysis of the respondents by various demographic characteristics is available in the Planning and Institutional Research Office. Next we examine those factors influencing student decisions to attend Mercy College.

Decision to Attend Mercy College

The three most mentioned reasons for attending Mercy College by the respondents were: (1) "Course Offerings"-16.5 percent (2) "Closeness to Home"-16.5 percent and (3) "Academic Reputation"-11.8 percent. Other reasons mentioned by at least five percent of the students include "a former students advice," "the availability of financial aid," "cost," "teacher/friends advice," and "counselors advice." This "advice factor" appears to be an advantage and coincides with findings of earlier studies by the College which suggest that once an individual participates in the Mercy experience, they leave satisfied.

An examination of the following table sheds light on how respondents learned about the College.

TABLE 27

How Respondents Learned About College

Source	Percentage
Relatives/friends	32.3
Material received in mail	18.9
College catalog	15.9
People at my high school	13.9
Representative of the college	7.2
Information display	4.0
Material in newspaper/magazine	2.9
Placement service	2.9
Other sources	1.6
Radio/TV advertisement	0.4

TABLE 28

Respondents Choice of Major by HEGIS Taxonomy

Program	Percentage
Business Administration	34.3
Undecided	15.8
Computer Information Sciences	12.5
Public Affairs & Social Services (includes CRJ, PSA, SOW)	7.8
Psychology	7.3
Social Sciences (includes HIS, POL, BSC, SOC)	5.2
Health Professions (includes Nursing and Medical Tech)	5.2
Letters (includes English and Speech)	4.3
Biological Science	3.6
Fine/Applied Arts	1.5
Foreign Languages	1.5
Mathematics	0.7
Communications	0.1

Males leaned towards programs in business, computer science, public affairs and social sciences. Women chose courses of study in all fields, concentrating in business, computer science, health professions and psychology. The number of undecided students divides equally by gender.

Most individuals responding to the survey enrolled on a full-time basis. Larger numbers of "Social Science" majors (eg. BSC, HIS, POL, SOC) were enrolled on a part-time basis than any of the other programs. This may be a function of program design and scheduling.

Several questions related to time and length of instruction were constructed for consideration. In response to a question about class meetings for the day session, students favored (53.5%) a sixteen-week course that meets twice a week for 1½ hours. Nearly one-fourth preferred a sixteen week term meeting once per week for three hours. About one-fifth preferred the schedule resembling the present eight-week term arrangement.

The same question asked about courses starting after 5:00 P.M. yielded slightly different results in terms of percentages. Greater numbers preferred the once per week format and the eight-week type courses. Still more than forty percent (41.9%) preferred the twice a week, sixteen week semester. This could be discounted by the Dobbs Ferry experience.

In response to the question, "What type of instruction do you prefer the most?" students chose lecture by instructor (41.2%), small-group discussion (38.7%) and self-paced instruction (18.9%). In later studies this question will be examined with respect to age as discussion regarding how adults learn continues.

Appendix B: Student Information Sheet

PLEASE USE INK AND PRINT NEATLY

COURSE _____ SPRING 1982

NAME _____

ADDRESS _____

PHONE _____

SS # _____

AGE _____

NUMBER OF COLLEGE CREDITS COMPLETED _____

NUMBER OF COLLEGE CREDITS TAKING THIS SEMESTER _____

COMPUTER COURSES TAKEN , WHERE TAKEN AND GRADES

1.

2.

3.

4.

5.

DO YOU WORK? _____ IF YES, BUSINESS PHONE _____ HOURS/WEEK _____

CAREER GOAL _____

SCHEDULE THIS SEMESTER FOR COURSES AND WORK-Block in hours

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday

NAME _____ PHONE _____

Appendix F: The Preliminary Programming Proficiency Test

Mat/Cis 134 Spring 1982 Pre-Test in BASIC

Name _____

This test in NO way influences your final grade. This test helps to determine the level and pace of the present programming course. This test refers to the BASIC language as taught at Mercy College on APPLE II.

-Here is a quick review (or an introduction) to some rules governing the syntax (grammar) of BASIC.

a)Each line of instruction is numbered and during program execution the line numbers are followed sequentially from lowest to highest.

b)Variable names are single or double letters or single letter followed by a digit (0-9).

c)Reserved words used for instruction statements are:

(1) 10 REM *()↑:.IJNBH$%%¢	REM is a remark statement and any characters may follow it.
(2)33 LET A=B-C	LET is an assignment statement. It perf.orms the calculation on the right-hand side of the equal sign and stores the answer under the variable name on the left-hand side of the equal sign. Symbols:addition(+),subtraction(-),division(/), multiplication(*),exponentiation(^).
(3)50 GO TO 998	GO TO is an unconditional transfer of the program from line number 50 to line number 998.
(4)88 IF A>B*C-34 THEN 100	IF...THEN is a conditional transfer of the program from line number 88 to line number 100 if the relation following IF is true. There are 6 possible relations that can be tested. =, >, <, >=, <=, <> (not equal).
(5) 25 INPUT D5,PP,X	INPUT takes in values from the keyboard for the variable names that appear in a list
(6) 56 PRINT X,PP,D4 56 ? X,PP,D4 30 ? "HELLO",X,Y	PRINT or ? types out the values from storage for the variables names in the list and any messages between quotes.

---

1 Treat each line as an individual line of BASIC code and determine if it is valid line of code or if it is an invalid line of code
For each write VALID or INVALID

(1) 20 LET C=5P+H/P (1)_____

(2) 30 ACCEPT X1,X2 ,X3 (2)_____

(3) 100 REM *"THIS IS INCORRECT* (3)_____

(4) 71 PRINT "THE ANSWER IS ",X (4)_____

(5) 80 LET A*B=3+DA3 (5)_____

(6) 90 INPUT 5.7,99 (6)_____

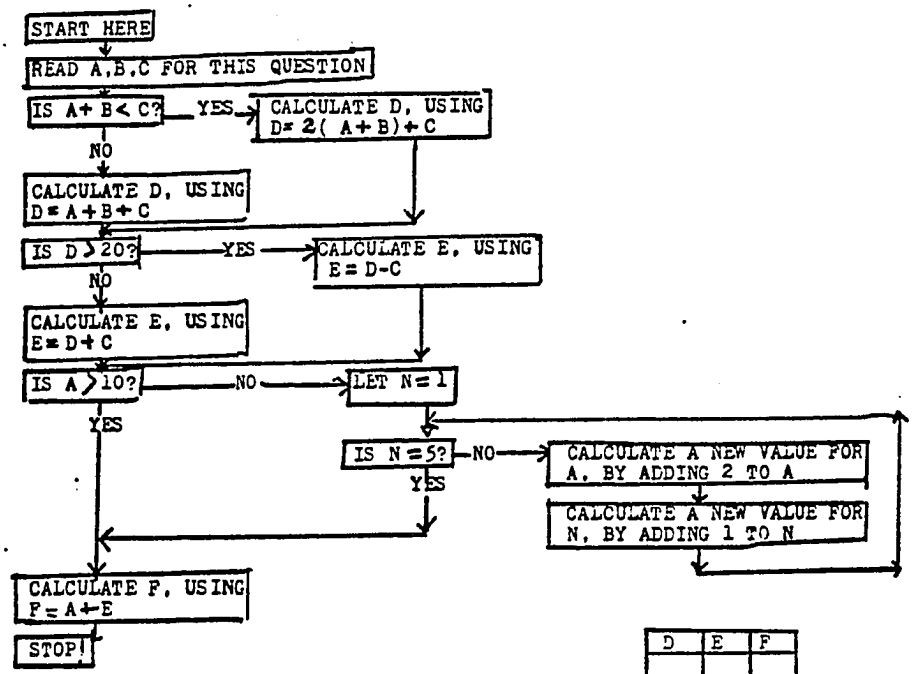
(7) 40 GO TO END (7)_____

(8) 100 IF A<>B THEN 89 (8)_____

(9) 90 LET B6=(X25+B2)*3 (9)_____

(10) 88 PRINT XYZ (10)_____

II Directions: Use the following diagram to answer the questions below.

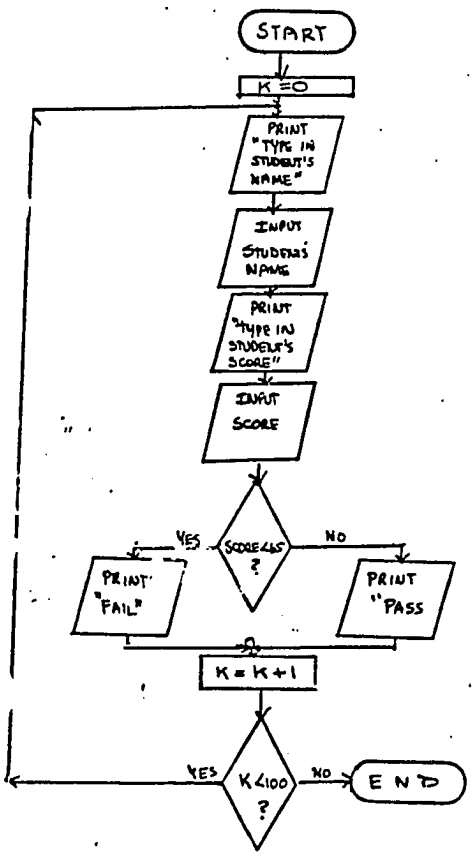


	D	E	F
(1)If A=2, B=3, C=6 , what is the final value of			
(2) If A=4, B=3, C=4, what is the final value of			
(3)If A=12, B=2, C=5, what is the final value of			

(4) Let A=5, and C=10. The value of B is unknown, but it is greater than 5. If the value of F is 25, what must value of B have been?
Choose one answer (a) 10
(b) 20
(c) 16
(d) 7
(e) none of these or impossible to determine.

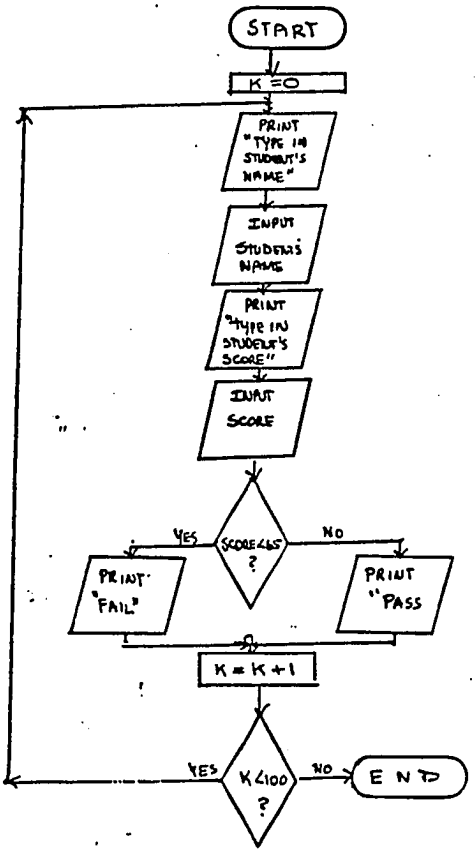
Part III- Below are the flowchart and the BASIC code for a program that processes the testscores of 100 students and determines whether they passed (>= 65) or failed the test.
You are to make changes in the flowchart or in the BASIC code so that the program will perform the modifications requested. Do one modification per diagram.

(A) Modify the program to print out a table of 3 columns with proper headings of NAME, SCORE, and GRADE



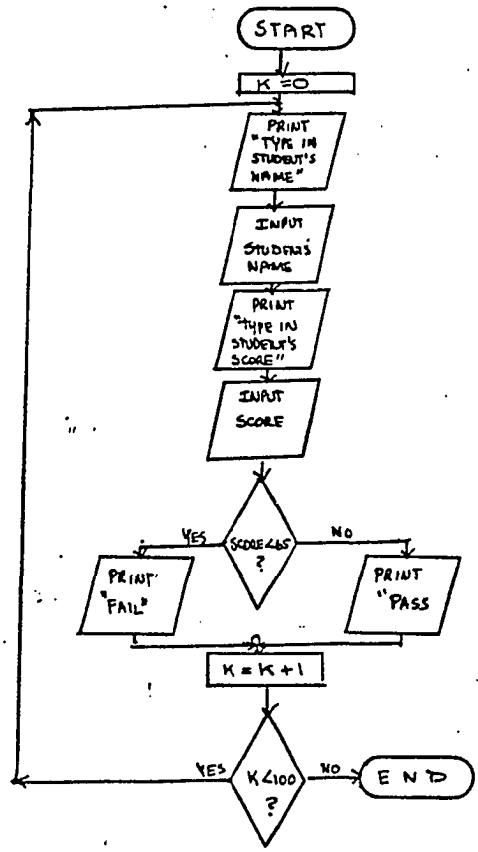

```
 100 LET K=0
 200 PRINT "TYPE IN STUDENT'S NAME"
 300 INPUT N$
 400 PRINT " TYPE IN STUDENT'S SCORE"
 500 INPUT S
 600 IF S 65  THEN 900
 700 PRINT"PASS"
 800 GO TO 1000
 900 PRINT "FAIL"
1000 LET K=K+1
1100 IF K 100 THEN 200
1200 END
```

(B) Modify the program to find and print the average of all the scores.



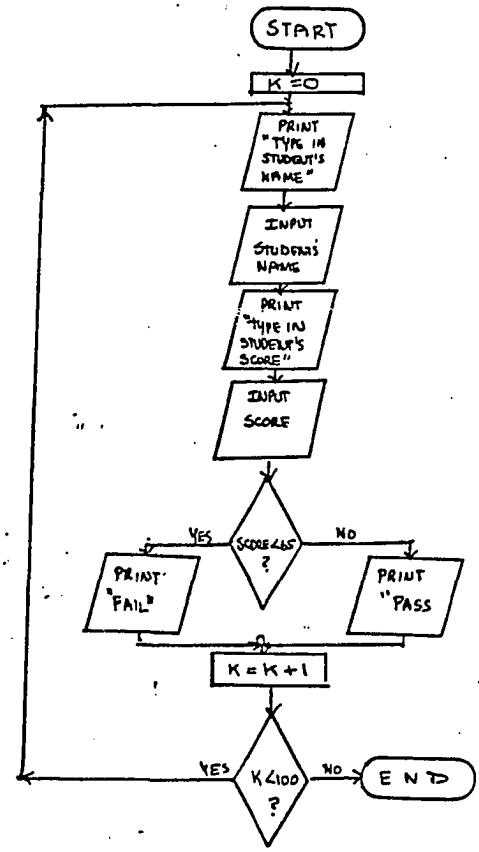

```
 100 LET K=0
 200 PRINT "TYPE IN STUDENT'S NAME"
 300 INPUT N$
 400 PRINT "TYPE IN STUDENT'S SCORE"
 500 INPUT S
 600 IF S 65 THEN 900
 700 PRINT "PASS"
 800 GOTO 1000
 900 PRINT "FAIL"
1000 LET K=K+1
1100 IF K 100 THEN 200
1200 END
```

(C) Modify the program to find and print the no. of students that passed and the no. of students that failed the test.




```
 100 LET K=0
 200 PRINT "TYPE IN STUDENT'S NAME"
 300 INPUT N$
 400 PRINT " TYPE IN STUDENT'S SCORE"
 500 INPUT S
 600 IF S 65  THEN 900
 700 PRINT"PASS"
 800 GO TO 1000
 900 PRINT "FAIL"
1000 LET K=K+1
1100 IF K 100 THEN 200
1200 END
```

(D) Modify the program to find the student with the highest score and print his/her name.




```
 100 LET K=0
 200 PRINT "TYPE IN STUDENT'S NAME"
 300 INPUT N$
 400 PRINT "TYPE IN STUDENT'S SCORE"
 500 INPUT S
 600 IF S 65 THEN 900
 700 PRINT "PASS"
 800 GOTO 1000
 900 PRINT "FAIL"
1000 LET K=K+1
1100 IF K 100 THEN 200
1200 END
```

(E) Describe, in words, how you would change this program if you didn't know in advance how many students there were to be processed.

Appendix G: BASIC Program to Form Three-member Heterogeneous Groups

```
10  PRINT "GROUP SELECTION"
20  PRINT "TYPE IN NAMES, GOING A
     CROSS THE ROW"
30  FOR K = 1 TO 6
40  INPUT GA$(K),GB$(K),GC$(K)
50  NEXT K
60  FOR J = 3 TO 7
70  PRINT "GROUPS FOR ASSIGNMENT"
     ;J: PRINT : PRINT
80  FOR JJ = 1 TO 6
90 DA$(JJ) = GA$(JJ)
100 DB$(JJ) = GB$(JJ)
110 DC$(JJ) = GC$(JJ)
120  NEXT JJ
130  FOR G = 1 TO 6
140 X =  INT (6 *  RND (1)) + 1
150  IF DA$(X) = "EMPTY" THEN 140

160 Y =  INT (6 *  RND (1)) + 1
170  IF DB$(Y) = "EMPTY" THEN 160

180 Z =  INT (6 *  RND (1)) + 1
190  IF DC$(Z) = "EMPTY" THEN 180

200  PRINT "GROUP";G
210  PRINT DA$(X),DB$(Y),DC$(Z): PRINT

220 DA$(X) = "EMPTY"
230 DB$(Y) = "EMPTY"
240 DC$(Z) = "EMPTY"
250  NEXT G
260  NEXT J
270  END
```

Appendix H: The Course Outline

CIS/MAT 13[illegible] STRUCTURED PROGRAMMING USING FORTRAN E. Grossman
Spring 1982

Individual Programming

Course Outline

Objective: It is assumed that students enrolled in this course have already been introduced to computers and programming. Structured Programming Using FORTRAN is intended to substanially develop the student's ability to employ good programming techniques such as structured constructs, top-down design and extensive documentation. The students will be taught to solve problems by forming algorithms using structured pseudocode and to translate them into the high-level programming language FORTRAN.

Text: Problem Solving and Structured Programming in FORTRAN; Friedman and Koffman; Addison Wesley. Second Edition

Grading: See attached page

Attendance: Attendance will be kept. It is the student's respnsibility to keep up with the work. We meet only once or twice a week. Get the phone numbers of several students NOW!

Office Hours: Dobbs Ferry- Mon.& Wed 12-1, Thurs.10-10:30
PSB- Phone 693-4500 EXT.298
Yorktown- Thurs. 11:30-12:00 & 3:00-3:30
Sat. 12-12:30
Faculty office. Phone (914) 245-6100

WEEK	MATERIAL COVERED/READINGS	ASSIGN. PROGRAM	ASSIGN. PROGR STAGE DUE IN CLASS
1	Login/Logout procedure,FORTRAN statement spacing, editing, review of computer systems, visit to terminal room. Chap.1	#1	
2	Statement types- TYPE,ACCEPT,FORMAT C,REAL,INTEGER, variable modes, conditional transfers, IF (p).... Chapts. 1,2	#2	#1 completed
3	Structured program concepts,flow-charts, algorithm development, pseudocode. Chapt. 3		
4	Library functions, arithmetic expressions Chapt. 4	#3	
5	Indexed DO loops Nested Do loops. Chapt. 4		#2 completed
6	One-dimensional arrays Chapt. 5	#4	
7	Arrays and implied DO Chapt. 5		#3 completed
8	Problem Session Chapts. 1-5 Midterm	#5	
9	Logical and character data. Case structure Chapters 6,9		#4 completed

WEEK	MATERIAL COVERED/READINGS	ASSIGN. PROGRAM	ASSIGN. PROG. STAG DUE IN CLASS
10	Formatting, Multidimensional arrays. Chapts. 8,10	#6	
11	Subprogramming- FUNCTION, SUBROUTINE, COMMON Chapt. 7		#5 completed
12	Problem session on SUBROUTINE and FUNCTION. Chapt.7	#7	#6 completed
13	File OPEN and CLOSE. In class programming project		#7 coded
14	Problem session. Chapts 6-10 Final Chapters 1-10		#7 first run
	At Dobbs Ferry campus on my desk by 4 P.M.		#7 completed!

A Note about your text:

Even the best programming texts make slow reading- give yourself plenty of time and don't read the assigned sections as if they were science fiction. One of the skills you must master for computer programming is meticulous, detailed thinking. To understand the examples, you will have to follow them almost letter by letter. You will find that such attention to detail is essential to write correct programs, and will save you time in the long run.

Grading: Grades are a necessary evil. Here is how I plan to grade your achievement in this course.
Midterm and Final will be 2 hour written exams. Both exams will be open book and similar to questions in exercises and discussed in class.
There will be 7 programming assignments. Each will be worth 10 points except the last one which will be worth 20 points. The grade for the programming assignment will be given on the final printed output- including a listing of the program and of the program output.
You will be responsible for understanding the reading material assigned up to date, the exercises and how your program works. The answers to each programming assignment will be available at the class meeting the week after the asssignment is due- if you do not hand in your assignments on time, you will be penalized. You are expected to hand in perfect assignments since they can be redone as often as you like before their due date.

Final Grade-			
Assigned Programs	20%	80 points	
Midterm	30%	120 points	
Final	50%	200 points	
Total	100%	400 points	

A 372-400 cumulative points - extral and original work
B+ 350 ≤ CP < 372
B 326 ≤ CP < 350
C+ 308 ≤ CP < 326
C 284 ≤ CP < 308
D 250 ≤ CP < 284
F CP < 250

Do your own programs. Any programs shared will cause a sharing of the grade.

Group Programming

CIS/MAT 134 COMPUTER PROGRAMMING I E. Grossman

Course Outline

Objective: It is assumed that students enrolled in this course have already been introduced to computers and programming. This course is intended to subsantially develop a student's ability to construct a structure algorithm using the high-level programming language FORTRAN. The students should also be instructed in good programming techniques by using structured constructs and extensive documentation. Students should be taught to design their own algorithms using structured pseudocode and then shown how this can be implemented in FORTRAN.

Texts: (A) Problem Solving and Structured Programming in FORTRAN: Friedman and Koffman : Addison Wesley.
(B)An Introduction to Structured Programming in FORTRAN: IBM GC-1790-0

Grading: Midterm and Final - 75% of final grade
7 assigned programs- 25% of final grade
Assigned programs are to be done in groups according to the schedule below.

Attendance: Attendance will be kept. It is the student's responsibility to keep up with the work.

Office hours: Dobbs Ferry- PSB#20- Tues :12-1:20,Thurs:12:50-1:20-6934500 ext.348
Yonkers- Faculty room- Thurs:11:50-12:20, Fri:12:00-1:00-963-0372

DATE	MATERIAL COVERED/READINGS	A.P.	A.P. STAGE DUE IN CLASS
	Login/Logout procedure, editing review of computer systems (A) chap.1	#1	
	Statement types- TYPE,ACCEPT, FORMAT, C, REAL ,INTEGER assignment, IF (p) GO TO... (A) 1,2 (B) 1	#2	#1 completed
	Structured program concepts, flowcharts, algorithm development pseudocode. (A) 3, (B)2,		#2 coded
	Library functions Arithmetic expressions (A) 4	#3	#2 1st run
	Indexed Do loop Nested Do loops (A) 4		#2 completed #3 coded
	One-dimensional arrays (A)5	#4	#3 1st run
	Arrays and implied DO		#3 completed #4 coded
	Problem session (A)1-5,(B)1,2 Midterm	#5	#4 1st run

WEEK	MATERIAL COVERED/READINGS	ASSIGN. PROGRAM	ASSIGN. PROG. STA DUE IN CLASS
10	Formatting, Multidimensional arrays. Chapts. 8,10	#6	
11	Subprogramming- FUNCTION, SUBROUTINE, COMMON Chapt. 7		#5 completed
12	Problem session on SUBROUTINE and FUNCTION. Chapt.7	#7	#6 completed
13	File OPEN and CLOSE. In class programming project		#7 coded
14	Problem session. Chapts 6-10 Final Chapters 1-10		#7 first run
To be announced	At Dobbs Ferry campus on my desk		#7 completed.

Appendix I: Group Programming Process

MAT/CIS 134

PROGRAMMING WITHIN A GROUP

Groups of 3-4 students will be formed to work on each programming assignment. The group will be fixed for that assignment and will meet three times during class time.

First class meeting: The group will analyze and discuss possible solutions for the assignment-sketching a flowchart- writing pseudocode.
During the following week: Each member will individually design a solution and write the code for the program and obtain a listing of the code - a copy for each member of the group. The copies will be left in a box in the terminal room with the other members' names on the copies. It is the responsibility of each member to pick up the listing of the other group member's program before the next class meeting.

Second Class Meeting: Each member will walkthrough the other member's listing (code) noting syntactical, logical and typographical errors. Each member will examine the program to see if it will process data properly - make up different values and do a desk check with them. Each member will return their copy of the listed code with comments to the author. Do not compare another member's program with yours but rather review it to determine if it will work- produce correct results. If the group dtermines the design is incorrect, the group should not attempt to redesign the program; rather the author should be directed to redesign the program and the group should make arrangements to reconvene at a later time.
During this week: Each member should remove the bugs that were noted and get a complete listing and run of the programming assignmant - again leaving copies for the other members in the terminal room. It is the responsibility of each member to pick up the listing and run before the next class meeting.

Third Class Meeting: Each member will walkthrough the other member's program listing and run, checking for good programming style and finding out why the program did or did not work. Each member will fill out comment sheets on the programs he/she reviewed during the walkthrough.
During this week: Each member is responsible to complete the assignment and hand in (1)a complete listing of program, run , printout
(2)comment sheets from group reviewers
(3)summary sheet on this assignment filled out by member.

You should find the class enjoyable and productive in terms of the time spent for what you learn. Your major resources are (1) team members (2) textbook/lecture notes (3) computer feedback and, of course, (4) the instructor.

Appendix J: Assigned Program Summary Sheet

MAT/CIS 134
ASSIGNED PROGRAM SUMMARY SHEET

Programmer's Name _____
Section_____
Assignment #_____

1. How much time (approximately) did you spend in designing an algorithm, translating it into code, debugging and rewriting this program? _____(in hours)

2. How much time did you spend in the terminal room to input and run this program? _____(printed by the computer at logout)

3. How many runs were necessary before this program "worked"?
_____(printed by computer as a generation number after your filename.
Example: EGR04.FOR;8 means
8 revisions/runs were done.

4. Briefly list the errors that were found on this program by the FORTRAN diagnostics- syntactical errors- at time of execution.

5. Briefly list the errors found in logic when the program would not run or output the correct results- logical errors.

6. Any additional comments on assigned program &/or programming in general.

Appendix K: Group Walkthrough Rating Sheet

MAT/CIS 134
EVALUATION FORM FOR WALK THROUGH IN GROUP

Author of Program_____

Reviewer _____

Assignment #_____

Section _____

Circle the response you choose

	No			Neutral or Don't Know			Yes
1. Were reasonable variable names used?	1	2	3	4	5	6	7
2. Were sufficient and useful comments provided?	1	2	3	4	5	6	7
3. Were spaces and blank lines used properly to produce a program with a pleasing format?	1	2	3	4	5	6	7
4. Was the logic of the program comprehensible. Could you follow what was being done in the program? Was this program easy to comprehend overall?	1	2	3	4	5	6	7
5. Was the algorithm a good choice? (ex Did the method of solution overlook some possible data choices; was it too complicated for such a simple problem?)	1	2	3	4	5	6	7
6. Would it be easy for you to modify this program if the original problem was altered slightly?	1	2	3	4	5	6	7
7. Would you have been proud to have written this program? Would you find it hard to improve this program?	1	2	3	4	5	6	7

Appendix L: Assigned Program Group Member Rating Sheet

MAT/CIS 134
ASSIGNED PROGRAM SUMMARY SHEET

Programmer's Name_____
Section_____
Assignment #_____

Circle the response you choose

	No		Neutral or Don't Know				Yes
1. Did you learn anything useful about the FORTRAN language during the walk-throughs with your group?	1	2	3	4	5	6	7
2. Did you learn anything useful about programming style during the group walk-throughs?	1	2	3	4	5	6	7
3. Do you modify your programming behavior to produce programs that are well-documented (full of comments, indentations) because they will be read by others?	1	2	3	4	5	6	7

4. How much time (approximately) did you spend in designing an algorithm, translating it into code, debugging and rewriting this program? _____ (in hours)

5. How much time did you spend in the terminal room to input and run this program? _____(printed by computer at logout)

6. How many run were necessary before this program "worked"?
_____(printed by computer as a generation number after your filename. Example: EX804.FOR;8 means 8 revisions/runs.

7. Briefly list the errors that were found on this program during the walk-through sessions
Syntactical Errors: Logical Errors:

8. Any additional comments on assigned program, group walk-throughs, &/or programming in general?

Appendix M: Midterm Examinations

NAME_____ Midterm FORTRAN MAT/CIS 134 Spring'82
FORM A

I. (A) Identify each of the following FORTRAN statements as valid or invalid. If it is invalid, identify the errors by rewriting the statement correctly.

(1) DATA KOUNT,KAT,KOOK/3*0/, A=8.2, B=-100.09

(2) DO K=1 TO 100 BY 1

(3) IF(.NOT.(A>ZERO) THEN A=B

(4) C=A**-BB

(5)C "HELLO THERE"

(6) X+Y = SQRT(A+B**2)

(7) FORMAT (1X,"I'M HERE TODAY")

(8) READ (X,Y,(J=1,100))

(B) Identify each of the following as valid or invalid FORTRAN variable names.

(1) MICKEY(MOUSE)_____ (4) 4LEAF _____

(2) SET-UP _____ (5) A2Z _____

(3) J.13 _____ (6) TROUBLE _____

(C) Identify each of the following as valid or invalid FORTRAN numeric input.

(1) 3,204 _____ (4) 321E3 _____

(2) 23/3 _____ (5) -43.008E-10 _____

(3) 400.94 _____ (6) $1.98 _____

(D) Translate the following formula (each single letter represents a variable) into FORTRAN assignment statement.

$$V = \frac{XYZ}{(A+B)^E} \quad \frac{X+Y+Z}{A-B}$$

I) Write a one-line FORTRAN statement that calculates the reciprocal of a value called NUMBER and rounds it correct to the nearest hundredth.
For example if NUMBER=9
Reciprocal is 1/9 = .111111 and your statement should produce .11
if NUMBER=8
Reciprocal is 1/8 = .125 and rounded to .13

II)

(a) What will be printed by this program in pseudocode using the following data : -3,3,15,15,24,8,999 ?

(b) Make up a pair of values for X and Y that will cause a problem on output in this program. Explain how and why.

```
READ X
WHILE X=999 DO
  READ Y
  IF X=Y
    THEN
      X=X**2
      Y=(X+Y)/2
      Z=200
    ELSE
      IF (X Y)
        THEN
          Y=Y**2
          Z=Y-X
        ELSE
          IF (X 0)
             THEN
               X=X/Y
          ENDIF
          Z=100
      ENDIF
    ENDIF
    PRINT X,Y,Z
    READ X
ENDWHILE
```

(III) Write a program in FORTRAN code using good programming style (as much as time permits) to:
(a) input an array of integer values less than 1000. The maximum size of the array is 100 and a value of 1000 will be used to indicate the end of input.
(b) search the array and find and print the value of the largest item in the array and its subscript.
(c) find and print the average of all the positive items in the array.

CIS/MAT 174 FORTRAN***MIDTERM Spring 1982 eg Form B ADA

Name :

Part I

(a)Identify each of the following FORTRAN staements as valid or invalid. If it is invalid then rewrite the statement correctly or identify clearly what the error(s) is.

(1) DO KITTY=1 TO KAT BY LITTER

(2)C "THIS IS WRONG -WHY?"

(3) IF(NOT(X<ZERO) THEN TYPE*, X

(4) ACCEPT*, THE, ENTIRE, ARRAY

(16)

(5) FORMAT(1X,"THIS IS JACK'S HOUSE")

(6) DATA (SALES(JAX),JAX=3,10,2)/5,7,0/

(7) A*B=C/D-E

(8) C=A*-B/KK

(b)Identify each of the following as valid or invalid FORTRAN variable names.

(1) INTERST_____ (2) B.123 _____

(6) (3) 7UP _____ (4) RISING(MOON)_____

(5) TEA42 _____ (6) SUM-UP _____

(c) Identify each of the folowing as valid or invalid FORTRAN numeric data input.

(1) 1234567 _____ (2) -50000.32 _____

(6) (3)$1.98 _____ (4) 3,864 _____

(5)-5.301E-55 _____ (6) 1/2 _____

(d) Translate the following formula (each single letter represents a a variable) into FORTRAN assignment statement.

(4)

$$X = \frac{A+B+C}{(MN)^2} - \frac{M+N}{ABC}$$

(E) Evaluate the following FORTRAN statement given the following values.

INK = ((A-B)/(C+D**E)*F)/G

A	B	C	D	E	F	G
25	3	2	3	2	4	2

_____ (4)

(F) Perform the following mixed mode arithmetic and write the final value of the expression as it is stored in variable named on left.

(1) P=4*(7/2) _____ (2) M=2.8*2 _____

(3) Q=FLOAT(4)+2.90+IFIX(-4.6/2) _____ (6)

(G) Write a one line FORTRAN statement that would round VALUE (positive, real) to the nearest thousandth. (Ex. if VALUE= 23.4888, it would become 23.489)

(4)

(H) If VALUE=123.456 , what would K equal after this FORTRAN statement is executed?

K=MOD(INT(VALUE),10)+MOD(INT(VALUE/10,10)+INT(VALUE)/100 (4)

(25)

II. a)Exactly what would be printed by the program below using the following data: 5, 8, 10, 10, 15, 5, 999
b)Make up a pair of data values for X and Y that this program will not be able to handle. Explain.

```
READ X
DOWHILE X ≠ 999
  READ Y
  IF(X=Y)
    THEN
     X=X**2
     Y=(X+Y)/2
    ELSE
     IF (X<Y)
       THEN
         Y=Y**2
         X=Y-X
       ELSE
         IF(X>0)
           THEN
            X=X/Y
         ENDIF
         Y=200
     ENDIF
  ENDIF
  PRINT X,Y
  READ X
ENDDO
```

(25)

III. Write a program in FORTRAN code using good programming style to:

(A)Input an array of non-zero real values. The array will be of maximum size 50. A value of zero will be used to indicate the end of input.
(B)Search the array to find and print the value of the smallest item and its subscript.
(C)Find the average of all the elements in the array and print it.

Appendix N: Final Examinations

MAT/CIS 134 -FORTRAN Final Spring 1982 Form A E. Grossman

NAME_____ Section_____

I. Determine whether or not each of the following statements are valid FORTRAN statements. If valid, write VALID. If incorrect, write a corrected statement.

(a) ARRAY(I,J)

(b) CALL PRTARR

(c) REAL SUBROUTINE SORT(A,NITMS)

(d) IF(X.OR.(Y.EQ.2)) GO TO 500

(e) FORMAT (1X, 5(A5,3x,"SAYA HELLO"/))

Given the following matrix, BETA, write the type and format statements needed to get the following output:

```
BETA
 1  2  3  4  5
 6  7  8  9 10
11 12 13 14 15
16 17 18 19 20
21 22 23 24 25
```

(f) _____

(g) _____

(h) Using a DO loop, write the statements necessary to print out the diagnol items of the matrix, BETA.

Tell whether each of the following logical compound expressions are TRUE or FALSE based on the values of the variables.

SIGNAL	FLAG	G	P	Q	A	FIRST	LAST
.TRUE.	.FALSE.	5.0	9.0	10.0	32.	'SAM'	'SYD'

(i) (G*P/Q.EQ.4.0).OR.(.NOT.(FIRST.GT.LAST)) _____

(j) SIGNAL.AND.(Q.GE.P.OR.FLAG) _____

(k) P.GT.Q.AND.Q.EQ.2*G.OR.Q.NE.A/5.AND.SIGNAL _____

(l) The following statements appear in a FORTRAN program after opening a data file:

```
      READ(44,99)IDNO,NAME,GPA,KLASS
99    FORMAT(I2,A4,F4.2,I5)
```

If you were preparing the data file that was to be read by these statements, how would you type in the following information to fit the format described above?

```
Identification number(IDNO)- 35
Class standing(KLASS)-       301
Name(NAME)-                  SAM
Grade Point Average(GPA)-    2.75
```

(25) Assume the values coming into the LIST array are 4,8,2,0,-1,10,3,999
Read the program and answer the questions below.

```
C     CALLING MAIN PROGRAM
      COMMON LIST
      INTEGER LIST(200)
      N=1
100   ACCEPT *,LIST(N)
      IF(LIST(N).EQ.999) GO TO 200
      N=N+1
      GO TO 100
200   CALL REVERS(N-1)
      TYPE*,(LIST(K),K=1,N-1)
      STOP
      END

      SUBROUTINE REVERS(SIZE)
      COMMON LIST
      INTEGER LIST(200), SIZE
      MIDDLE=SIZE/2
      DO 100 I =1,MIDDLE
          CALL SWAP(LIST(I),LIST(SIZE+1-I))
100   CONTINUE
      RETURN END

      SUBROUTINE SWAP(K,J)
      INTEGER K,J,AUX
      AUX=K
      K=J
      J=AUX
      RETURN
      END
```

(a) What would be printed at the end of the calling program?

(b) Why was COMMON not needed in SUBROUTINE SWAP?

(c) What is the relation between LIST in the subroutine REVERS and in the calling program?

(d) What is the relation between K in the subroutine SWAP and K in the calling program?

(e) What is the relation between the statement 100 in the calling program and the statement 100 in subroutine REVERS?

(f) How many times is the subroutine SWAP executed during this run?

Choose (A) OR (B) (40)

Write answer on back of test paper or use own paper- Make sure your name is on every paper you hand in.

(A)Represent the cards of a bridge hand by a pair of integers as done in the class problem poker. Use a 4X14 matrix and leave the first column blank (full of zeros) and let the ace be the fourteenth column.
(1)Deal a hand of 13 cards.
(2)Compute the number of points in the hand by scoring a 4 for each ace
a 3 for each king
a 2 for each queen
a 1 for each jack.

Do this in a subroutine.
(3)In a subprogram function add 3 to the accumulated points for each suit not present in the hand. Return the total points to the main program where it will be printed.

EX: 2♠, 4♡, 4♡, 6♠, 7♡, 10♡, 8♠, J♡, Q♠, Q♡, K♠, 5♠, Ace♡.
Face value points - 12
Missing suit points - 3 (2 missing)
Total points - 15

t

Use a minimal of comment statements We are forced to compromise good programming style in order to perform will in time and space allowed.

(6) Mr. Klaus is quite busy at this time of the year taking a toy inventory at the North Pole. He has a list of his 5 elf divisions and the amount to toys each division produced for each month thisyear.

	J	F	M	A	MY	JE	JY	AT	S	O	N	D
ELF DIVISION 1	234	122	150	361	714	801	265	717	811	107	125	144
ELF DIVISION 2	486	354	112	403								
ELF DIVISION 3	921											
ELF DIVISION 4	101											
ELF DIVISION 5	88											

a)Help him by loading (input)this matrix - be sure to give specific directions on how data is to be typed.
b)Using nested DO loops calculate the average number of toys produced each month and put them into an array (1-dimension).
c) Write a subroutine to arrange the average array in descending order .
D) In the main program print the number of the month that has the highest average - (Next year the elves will all get a vacation during this month)
E) How would you modify this program to find the total number of toys in the inventory?

MAT/CIS 134 FORTRAN Final- Form B Spring 1982 E. Grossman
Name_____Section_____

I. Determine whether or not each ofthe following statements are valid FORTRAN statements. If valid, write VALID. If incorrect, write a corrected statement.

(a) FORMAT (1X, 3(A5,3x "IS THE ANSWER"//))

(b) ARRAY(R,C)

(c) CALL PRTERR

(d) IF (X.EQ. 1.OR.2) GO TO 100

(e) INTEGER FUNCTION (LIST,SIZE)

II. Given the following matrix, BETA, write the type and format statements needed to get the following output.

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

(f) ØØ3Ø84
ØØ8ØØ9
Ø13Ø14

(g) Ø18Ø14ØØ7ØØ1ØØ16ØØ3

III. (h) Consider the following instruction:

DATA ((ALPHA(R,C),C=1,3),R=1,2/4,5,6,7,8,9/

How would this data be stored? Draw a picture of the matrix with the data stored in it.

IV. Tell whether each of the following logical compound expressions are TRUE or FALSE based on the values of the variables.

A	B	C	I	J	K	PINK	RED	NAME1	NAME2
1.0	2.0	3.	5	10	0	.TRUE.	.FALSE.	'JOE'	'JON'

(i) (A*B/C.EQ.K).OR..NOT.(NAME1.GT.NAME2) _____

(j) PINK.AND.(C.GT.B.OR.RED) _____

(k) A.GT.B.AND.B.EQ.2*A.OR.K.NE.J/10.AND.PINK _____

V. (l) The following statements appear in a FORTRAN program after opening a data file:

```
      READ(44,99)IDNO,NAME,GPA,KLASS
99    FORMAT(I2,A4,F4.2,I5)
```

If you were preparing the data file that was to be read by these statements, how would you type in the following information to fit the format described above?

Identification number(IDNO)- 35
Class standing(KLASS)- 301
Name(NAME)- SAM
Grade Point Average(GPA)- 2.75

II

(25) Assume the values coming into the LIST array are 4,8,2,0,-1,10,3,999
Read the program and answer the questions below.

```
C     CALLING MAIN PROGRAM
      COMMON LIST
      INTEGER LIST(200)
      N=1
100   ACCEPT *,LIST(N)
      IF(LIST(N).EQ.999) GO TO 200
      N=N+1
      GO TO 100
200   CALL REVERS(N-1)
      TYPE*,(LIST(K),K=1,N-1)
      STOP
      END

      SUBROUTINE REVERS(SIZE)
      COMMON LIST
      INTEGER LIST(200), SIZE
      MIDDLE=SIZE/2
      DO 100 I=1,MIDDLE
         CALL SWAP(LIST(I),LIST(SIZE+1-I))
100   CONTINUE
      RETURN END

      SUBROUTINE SWAP(K,J)
      INTEGER K,J,AUX
      AUX=K
      K=J
      J=AUX
      RETURN
      END
```

(a)What would be printed at the end of the calling program?

_____

(b)Why was COMMON not needed in SUBROUTINE SWAP?

_____

(c) What is the relation between LIST in the subroutine REVERS and in the calling program?

_____

(d)What is the relation between K in the subroutine SWAP and K in the calling program?

_____

(e)What is the relation between the statement 100 in the calling program and the statement 100 in subroutine REVERS?

_____

(f)How many times is the subroutine SWAP executed during this run?

III

Choose (A) OR (B) (40)

Write answer on back of test paper or use own paper- Make sure your name is on every paper you hand in.

(A)Represent the cards of a bridge hand by a pair of integers as done in the class problem poker. Use a 4X14 matrix and leave the first column blank (full of zeros) and let the ace be the fourteenth column.
(1)Deal a hand of 13 cards.
(2)Compute the number of points in the hand by scoring a 4 for each ace
a 3 for each king
a 2 for each queen
a 1 for each jack.
Do this in a subroutine.
(3)In a subprogram function add 3 to the accumulated points for each suit not present in the hand. Return the total points to the main program where it will be printed.

EX: 2♦, 4♣, 4♠, 6♥, 7♣, 10♣, 8♥, J♠, Q♦, Q♣, K♦, 5♦, Ace♣.
Face value points - 12
Missing suit points - 3 (♥ missing)
Total points - 15

t

Use a minimal of comment statements We are forced to compromise good programming style in order to perform will in time and space allowed.

(6) Mr. Klaus is quite busy at this time of the year taking a toy inventory at the North Pole. He has a list of his 5 elf divisions and the amount to toys each division produced for each month thisyear.

	J	F	M	A	MY	JE	JY	AT	S	O	N	D
ELF DIVISION 1	234	122	150	361	72	801	205	77	811	109	123	144
ELF DIVISION 2	486	364	112	403				12				
ELF DIVISION 3	92											
ELF DIVISION 4	101											
ELF DIVISION 5	86											

a)Help him by loading (input)this matrix - be sure to give specific directions on how data is to be typed.
b)Using nested DO loops calculate the average number of toys produced by each elf division and put them into a one-dimensional array.
c)Write a subroutine to arrange the average array in ascending order.
d)In the main program print the number of the elf division that has the highest average . (Next year these elves will ride in the sled)
e)How would you modify this program to find the total number of toys in the inventory?

Appendix O: Programming Assignments

CIS/MAT 134 ASSIGNMENT #1

Consider the following program:

```
00100 C         YOUR NAME      ASSIGNMENT #1
00200 C
00300 C DECLARATION STATEMENT
00400         REAL HOURS, RATE, TAX, GROSS, NET
00500 C
00600 C FORMAT STATEMENTS
00700 00100   FORMAT (1X,'PLEASE ENTER NUMBER OF HOURS WORKED')
00800 00200   FORMAT (1X,'PLEASE ENTER PAY RATE')
00900 C
01000 C DATA INITIALIZATION
01100         DATA TAX/25.00/
01200 C
01300 C INPUT DATA
01400         TYPE 100
01500         ACCEPT *,HOURS
01600         TYPE 200
01700         ACCEPT *,RATE
01800 C
01900 C CALCULATE GROSS PAY
02000         GROSS = HOURS * RATE
02100         NET = GROSS - TAX
02200 C
02300 C OUTPUT RESULTS
02400         TYPE *,GROSS
02500         TYPE *,NET
02600 C
02700         STOP
02800         END
```

Using the program above, perform the following procedures. Clearly indicate each step on your printout. Refer to your notes on terminal usage while doing this.

1. LOGIN
2. CREATE a source file for your program.
   (use FILLN.FOR for your filename)
3. Enter your program lines.
   (correct any typing errors using CTRL/U or DEL)
4. EXECUTE your program.
5. EDIT your program.
   Make the following changes while in EDIT mode.
   (A) Print line 00700
   (B) Print lines 01400 to 01700
   (C) Delete line 02000
   (D) Change line 02100 to
   ```
   02100  NET = (HOURS * RATE) - TAX
   ```
   (use S command)
   (E) Replace line 02400 by
   ```
   02400  TYPE 300
   ```
   (F) Replace line 02500 by
   ```
   02500  TYPE *, HOURS, RATE, NET
   ```
   (G) Insert line 00850
   ```
   00850 FORMAT (1X,'HOURS',15X,'RATE',16X,' NET  PAY')
   ```
   (H) Save the new file
6. TYPE out your new program.
7. EXECUTE your new program.
8. LOGOUT.

MAT/CIS 134
-.1782
E. Grossman

Assignment #2

Check outline for due date.

Text page 39/problem 1D with some additional requirements:

Write a program,using good programming style, to input three data items into variables X,Y, and Z and find and print with the proper identifying comments their
a)product
b)sum
c)the largest of the three values. Hint: Use an additional variable named LARGE which should always contain the largest of the data items checked so far.

Test your program with the following data:
(It will be necessary to run-EXECUTE- the program 5 times- once for each set of data)

(a)	(b)	(c)	(d)	(e)
5	0	45	-1	1
5	6.77	6,054.3	-.05	-.05
5	6	1/3	-2.1E+5	-.002

MAT/CIS 134 ASSIGNMENT#3 EG

123/3H

Write a program to read in a collection of exam scores ranging in value from 1 to 100, use an. appropriate sentinel value to indicate the end of input, and count and print the total number of scores, and the number of outstanding scores (90-100), the number of satisfactory scores (60-89) and the number of unsatisfactory scores (1-59). Test your program with the data in the text and don't forget to use an endcode.
Also calculate and print the average exam score.

This program should employ DOWHILE and IFTHENELSE structures, 'creaming off the top' procedure and a cumulative sum.

Let's write it in pseudocode now.....

MAT/CIS 134 ASSIGNMENT #4 E. Grossman

178/4L

Run this program twice - once with the data in text and again with the data given below.
Note: For each run be sure to print out a request to read in a value indicating how many lines of data there are.

Employ library functions and the DOUNTIL structure (FORTRAN code- DO loop). Round off to the nearest penny. Make sure to have data in neat tabular form with columns & headings of table clearly worded, for output.

Second set of data (for second run):

Loan in $	Months	Rate in %
1,000,000	12	18
7,500	60	11.1
100	120	13.26

MAT/CIS 134 ASSIGNMENT #5 E. Grossman

Text: Page 231/ 5M

This is a problem that could be broken down into several parts - each part is 'attacked' separately and then united when each part is correct.

Part (A) Input 3 arrays using a DOWHILE - (since a sentinel value is used) and output a neat, clearly headed table of three columns using a DOUNTIL structure. (-If you counted the number of households as they came in)

Part (B) Calculate the average household income using DOUNTIL and cumulative sum.

Part (C) Search the household income array for all that exceed the average household income and print them out in neat tabular form with proper heading (Entitle the table and then have two columns headed - ID NO INCOME etc.)

Part (D) Determine the percentage of households below poverty income - search the array testing each income to see if it is below the poverty level (using the formula in text) and then:

$$\text{percentage} = \frac{\#\text{ of households of income } \overset{\text{below}}{\wedge} \text{ poverty}}{\text{total \# of households}} * 100$$

MAT/CIS 134 ASSIGNMENT #6

Each person employed by the Terrific Terminal for Today company is strongly encouraged to meet a minimum level of sales of $10,000 in terminal systems each month. The person who meets this minimum level is paid a base salary each month of $1000 plus a coded percent commission on all sales exceeding $10000. If the code for the commission is 1 - that salesperson receives 3% commission ; if the code is 2 that salesperson receives 5% commission.
Those who do not meet the minimum in sales are paid $300 plus 2% of his/her sales for the month.
The outstanding sales persons who sell more than $15,000 worth of terminal systems are paid an additional bonus of 10% of all their sales over $15,000.

The data for each salesperson will consist of a 5-letter name, the amount of that month's sales and a commission code.

Put the names into a NAME array and the corresoponding sales and code into a SALES array and CODE array. Compute the monthly salary for each salesperson using multiple decision-alternative structure (IN CASE) and logical compound expressions. Put the salaries into a corresponding SALARY array. Use a SORT routine on the salaries and put them into ascending order (lowest first). Don't forget to move the NAME array SALES array and SALARY array in the sort.

Output a 3-column table with the results using proper headings and listing the salaries in the table in ascending order.

SALESPERSON SALES SALARY

Use the following data and make up 5 more names with sales and codes

Name	SALES	CODE
SMITH	14999.99	1
JONES	9000.00	2
COOP	456.78	2
HAY	23456.78	1
LOVER	11111.11	1

Declare names to be integer mode and limit them to 5 characters.

MAT/CIS 134 PROGRAMMING ASSIGNMENT 7 EG

A TEAM Programming Project: Subprogramming

Your team should break the program below down into modules and each member in the team should take one or more modules to code and test.

During this week ( ) code your module(s) and leave a listing of the code in the terminal room - a copy for each member of your group (put their name on their copy). Also pick up the listing of code of other modules done by the other team members (with your name written on the copy). Read and critique each listing and when the class meets ( ) you can meet with your team and exchange ideas and help each other debug each module before you attempt to run them.

During the next week ( ) put your corrected, coded module(s) on file in the computer , make up data and test the module. You will probably have to add lines to test your module(s) and later you will have to delete those lines and prehaps add other lines of instruction in order to run the module(s) with the actual data.

Next time the class meets ( ) bring in the runs of your module(s) with any problems to the team exchange and critique each. Run the entire program during the next week ( ) linking each member's module by using the command: EXECUTE PROG1.FOR,PROG2.FOR, etc.

Actual data for the update will be on file to be specified later.

1981-1982 COLLEGE ENROLLMENT

		SESSION		
		FALL	SPRING	SUMMER
C	DB	3578	3765	1988
A	YT	1456	1768	980
M	YN	1896	2005	1143
P	WP	658	567	333
V	PK	456	345	202
S	BX	758	657	394

Here is a matrix table displaying how many students are attending each campus for each session at College for the academic year. During the year this table needs a constant update as students register and students drop out.

Program Modules:

A) Main calling program - introduction and description of program and all variable names used; organize calls and references to subprograms with proper mode and order of arguments.
B) Subroutine- INPUT - the data from enrollment table - include instructions on how to type in data.
C) Subroutine - DISPLAY - the matrix table of enrollment.
D) Subroutine - Input NEW data - from a data file and updatr matrix. Include check for data errors.
E) Function - Calculate the number of students on each campus. Type those values with appropriate identification in the subprogram and return to the main program the total number of students attending by campus.
F) Function - Calculate the number of students attending each session. Type those values in subprogram with appropriate identification and return to the main program the total of students attending by session.
G) Main program should print out the total enrollment results from (E) and (F) and make sure that they are the same.
H) After Update is completed (D), display the new matrix.

Appendix P: Attitude Toward Computers and Computer Programming Survey

MAT/CIS 134

COMPUTER ATTITUDE SURVEY

Section_____

Please check the box that most accurately reflects how you feel about the statements to the left.

	STRONGLY DISAGREE	DISAGREE	NEITHER AGREE NOR DISAGREE	AGREE	STRONGLY AGREE
1. Computer programming is an important skill to have for future job opportunities.					
2. I try to avoid working on my computer programs until the last minute. They are my least favorite assignments.					
3. Computers are <u>not</u> very useful because they are always breaking down or making mistakes.					
4. Computers are fascinating and exciting to work with.					
5. I become frightened and panicky at the thought of having to write a computer program.					
6. I will try to avoid working with computers in my future job.					
7. Computers will help to bring about a better way of life for the average person.					
8. Computer programming seems to be a fairly useless skill.					
9. Computers are extremely accurate, efficient and reliable.					
10. I always look forward to working on my computer programming assignments.					
11. Computers are important for the efficient operation of large and small businesses.					
12. I hestitate working with computers because they are strange and anxiety-provoking.					
13. I find that discovering solutions to programming problems is a logical process.					
14. Every college student should be required to take a course in computer programming.					

	STRONGLY DISAGREE	DISAGREE	NEITHER AGREE NOR DISAGREE	AGREE	STRONGLY AGREE
15. I view programming as a stimulating and challenging activity.					
16. I find that trying to get my program to run on the computer is a frustrating ordeal.					
17. It seems to me that a programming class motivates students to cheat - they copy other people's programs and hand them in.					
18. One cannot use what is taught in this course outside of a programming environment.					