A COMPARISON OF THE ABILITY OF NOVICES AND EXPERIENCED THIRD GENERATION LANGUAGE PROGRAMMERS TO LEARN FOURTH GENERATION LANGUAGES

by

CHARLES E. PULFER

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE IN BUSINESS ADMINISTRATION

in

THE FACULTY OF GRADUATE STUDIES

Commerce and Business Administration

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

March 1987

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the The University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Commerce and Business Administration

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date: March 1987

ABSTRACT

This thesis describes research which was carried out to determine whether novices could program in fourth generation languages as well as experienced third generation programmers.

It was thought that experience with a third generation language could be transferred to a fourth generation environment. This hypothesis was tested using a completely randomized block design lab experiment consisting of two factors and a block. The two factors were experience with third generation languages, and complexity of the task. The block was the educational institution where the lab sessions were conducted. Each of the factors and the block had two levels. The specific hypotheses tested were:

1. Experienced third generation language programmers will record higher mean scores on both simple and complex tests of fourth generation languages.
2. The difference in test scores, between simple and complex fourth generation language tasks, will be greater for novices than for experienced third generation language programmers.
3. Experience with other software tools, especially report writers, query languages, and other fourth generation languages will affect the subjects' performance on the fourth generation language tests.

Using FOCUS as the fourth generation language, lab sessions were run for fifty-seven subjects. The results indicate that experience with third generation languages affects a subject's performance on simple tests of fourth generation languages. The results also indicate that the experience has no effect on

complex tests of fourth generation languages. Because of a lack of data, no meaningful conclusions could be reached for hypothesis number three.

We feel experienced third generation language programmers scored higher than novices on simple 4GL reporting tests because experienced 3GL programmers had skills which were very similar to the skills needed in a simple 4GL reporting application.

There are several possible ways of explaining why experienced programmers could do no better than novices on complex 4GL reporting tests. One possible explanation follows; because complex 4GL reporting commands are so different from third generation language commands, third generation language programmers had no advantage over novices. A second explanation might be that the complex test was too difficult, or too long. As a result of this difficulty, no one was able to perform very well.

We conclude that experienced programmers should be preferred over novices when applications involve simple 4GL commands. More research is necessary to determine if in fact novices can perform as well as experienced third generation language programmers on complex 4GL tasks.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

This research was supported by an NSERC Postgraduate Scholarship awarded to Charles Pulfer in 1985.

1. INTRODUCTION

This research was prompted by the growing acceptance of a new type of software in the corporate information systems environment. Though these "fourth generation languages" are growing in acceptance (in 1985 a study found that fourteen percent of IBM installations in the U.S. use fourth generation languages.)[1] , information systems managers still know little about the realities of this type of software. Information systems managers have trouble defining the term "fourth generation language." Their inability to define the term is caused by the fact that software vendors, marketing everything from report writers to database management systems, label their products fourth generation languages (4GL's). As well, these software vendors claim that their products can not only be used by computer novices, but also by experienced programmers. If information systems managers are to make proper use of fourth generation languages, they will need a clearer indication of what they are, who can use them, and when they should be used. The research was begun with these aims in mind.

The purpose of this experiment was to provide insight into the ability of novices and experienced third generation language programmers to learn a fourth generation language. The primary goal was to determine whether knowledge of third generation languages affected a person's ability to learn a fourth generation language. A secondary goal was to determine whether novices had more difficulty with complex fourth generation language commands than did experienced third generation language programmers. It was hoped that answers to these questions would help information systems managers decide

---

[1] "4GLs enter dp mainstream despite some resistance", Computing Canada: Software Report, (May 1985), p.6.

1. who should do the programming in a fourth generation environment, and
2. whether novices should be allowed to produce more complicated applications, or whether this task should stay within the information systems department.

The rest of the thesis proceeds as follows. Chapter 2 reviews the literature on fourth generation languages, Chapter 3 reviews some prior research relevant to fourth generation languages, Chapter 4 reviews some relevant theory on learning, Chapter 5 describes the method used for the experiment, Chapter 6 analyses the data obtained from the experiment, and Chapter 7 discusses the results.

2. REVIEW OF THE LITERATURE ON FOURTH GENERATION LANGUAGES

2.1 DEFINITION OF FOURTH GENERATION LANGUAGES

As previously mentioned, one of the biggest problems involved with doing research in this area is the lack of a precise definition of these languages. Without a precise definition, information systems managers are not equipped to handle the combination of marketing literature, and "buzzwords" produced in the practitioner literature. For this reason, the first step in this thesis was to uncover the characteristics which define a fourth generation language.

Martin[2] defines a fourth generation language as a tool which will result in a productivity improvement of at least ten to one over COBOL. Fourth generation languages also use an order of magnitude fewer lines of code, when developing an application, than would be needed with COBOL, PL/1 etc. Therefore fourth generation languages might be characterized as high productivity languages. The claimed productivity improvement is a result of the languages using a diversity of other mechanisms, besides sequential commands, such as filling in forms or panels, screen interaction and built-in defaults. Unfortunately little research has been done to substantiate the productivity improvement claims put forward by vendors. Therefore, we cannot be sure that fourth generation languages increase productivity.

In addition, we cannot be sure if a fourth generation language can be used for all applications, or whether it is only suitable for a certain core of applications. Third generation languages such as COBOL and FORTRAN are domain independent. They are used across a

---

[2]James Martin, Application Development Without Programmers (Toronto: Prentice Hall,1982), p.28.

variety of application areas and do not incorporate domain specific knowledge. Some very high level languages (e.g. IFPS, GPSS) are domain dependent. That is to say, they can only be applied to solve specific problems. Fourth generation languages vary greatly in their power and capabilities. While a third generation language could create all or most applications, some fourth generation languages are designed only for a specific class, or range of applications. Some are highly restricted in their range, while others can handle a diversity of applications well. In some cases, a specific fourth generation language might have to be chosen for a specific application. On the other hand, some 4GL's are just as flexible as COBOL and can be used to produce complete applications in almost any area of business.

Fourth generation languages are also characterized as problem oriented, [3] or nonprocedural. As Leavenworth and Sammet state :"It is hard to convey an intuitive notion of languages which in some sense are higher than FORTRAN, COBOL, PL/1 etc. The most common term used for this concept has been nonprocedural, and the most common phrase has been 'what' rather than 'how'. That phrase refers to the facility of a user to indicate the goals (what) he wishes to achieve rather than the specific methods of solution (how) that must be used." [4] In discussing the advantages of a nonprocedural language Leavenworth and Sammet state: "The solution should be specified implicitly in terms of structures or abstractions which are relevant to the problem rather than those operations, data and control structures which are convenient for some machine organization."[Ibid, p.2.]

---

[3] Steven L. Mandell, Computers and Data Processing: Concepts and Applications (New York: West Publishing,1985), pp.246-247.

[4] Burt M. Leavenworth and Jean E. Sammet, "An Overview of Nonprocedural Languages", IBM Research Report RC4685 (1974), p.1.

Leavenworth and Sammet also identify some characteristics of these nonprocedural or problem oriented languages

1) Associative referencing - the programmer does not have to specify access paths, or conduct a search for a specific data structure. The data is accessed on some intrinsic property of the data.

2) Aggregate operators - no need for looping.

3) Elimination of arbitrary sequencing - If a program satisfies the "single assignment" test (no variable is assigned values by more than one statement) then the order of the statements is immaterial.

4) Pattern directed structures - search for a pattern without specifying how to search.

The degree of nonproceduralness of a language is not absolute, but, rather, is relative. A third generation language with a statement such as A=(B * C)+D can be considered nonprocedural when compared to the equivalent operation in assembly language. Generally, we can state that a fourth generation language is more nonprocedural than a third generation language. Some of the fourth generation commands can be expressed in terms of a series of third generation commands. But, as Elder states: "A fourth generation language that is entirely nonprocedural will allow users to retrieve information without detailed programming but will be limited to queries only. To develop applications that involve any logical decisions and/or the processing of data (e.g. sorting) a language must have procedural aspects.... The difference between a procedural fourth generation language and third generation languages is the number of procedural instructions necessary to write an application."[5]

---

[5] Marvin Elder, "SALVO - A Fourth Generation Language for Personal Computers", Proceedings of the National Computer Conference 1984

Most fourth generation languages contain procedural commands, for example IF's and GOTO's, in order to handle more complex logic. Schmidt attempts to build database querying capabilities into PASCAL.[6] We can see that the PASCAL commands are substantially longer than the equivalent commands in IBM's SQL, or a fourth generation language. This gives an indication of the differences between a fourth generation language and a third generation language. The savings realized in lines of code by a 4GL are usually a result of more powerful nonprocedural commands incorporated into a 4GL. Savings are also realized as a result of default options chosen by a 4GL. Most 4GLs can automatically select the format of a report, put page numbers on it, select chart types for graphic display, put labels on the axes or on column headings, and ask the user in a friendly manner when it needs more information.

Jenkins [7] has classified current software generator products into three classes: 1) application generators, 2) code generators, and 3) productivity enhancement tools. Application generators have an end user orientation. Application generators will produce complete working applications, such as payroll or accounts receivable, from specifications given by the user. FOCUS and RAMIS II are examples. Code generators produce a coded program, in third generation language, from the given specifications. This coded program would then have to be compiled and run. These code generators are oriented more towards technical users who can "fine tune" the code. Productivity enhancement

---

[5](cont d) (Montvale,N.J.: AFIPS,1984), p.564.
[6]Joachim W. Schmidt, "Some High Level Constructs for Data of the Type Relation", ACM Transactions on Data Bases, 2(1977), pp.247-261.
[7]Milton A. Jenkins, "Surveying the Software Generator Market", Datamation, 31, No.17(1985), pp.247-261.

tools facilitate the development process but cannot produce whole applications by themselves (examples of productivity enhancement tools include query languages, test data generators, automatic documentation, and report generators.) Both application generators and code generators fall into the category of fourth generation languages, but productivity enhancement tools are not considered 4GL's because they can produce only a portion of the total application.

Schussel outlined three types of fourth generation languages in a slightly different manner. [8]

1) Interpretive programming languages are nonprocedural and easy to learn. Examples include NOMAD2, RAMIS II, FOCUS and NATURAL. These languages do not handle complex logic well.

2) Function generators interact with the developer in the form of dialogues and screens. Examples are ADS/O, MANTIS and IDEAL. As an example, an IDEAL application consists of two major classes of components:

1. Fill-in-the-blank screens for description of the application: its inputs and outputs, reports, and panels.
2. A high level procedural language incorporating relational commands. With this system, the developer is insulated from the operating system, teleprocessing monitor, and data base manipulation.

3) Compiled system generators will generate COBOL or PL/1 code from high level specifications. Examples include PacBase, GAMMA, TELON, and UMBRELLA.

---

[8] Schussel, the President of Digital Consulting is quoted in the following article, Miriam Cu-Uy-Gam, "Do-it-yourself is on the way for system development", Computing Canada: Software Report, (May 1985), p.9.

Schussel's compiled system generator category is equivalent to Jenkins' code generator category. Both function generators and interpretive programming languages fall into Jenkins' application generator category. Table 1 summarizes Jenkins' and Schussel's categorizations.

Jenkins' term "application generators" is used in a business environment as a synonym for a fourth generation language. Grochow provides an introduction and bibliography on the topic of application generators.[9]

Fourth generation languages use database management systems to improve productivity. Fourth generation languages such as FOCUS, NOMAD or MAPPER are built on top of, and integrated with, their own database management system. Other 4GL's can be joined to database management systems already in place. One of the keys to efficient application generation is the fact that the data structure for an application already exists; it is represented in the database data dictionary which the software can use. The creator of an application is not required to design the data or its structuring.

The data dictionary is the foundation of many report generators, query languages and application generators. In addition to describing the data, such dictionaries may contain report headings, alternate names for data (aliases), report formats, screen layouts, and titles for fields that can be placed in column headings.

Fourth generation languages suitable for users, as well as computer professionals, are emerging from three primary sources:

---

[9] Jerrold M. Grochow, "Application Generators: An Introduction", Proceedings of the National Computer Conference 1982 (Montvale,N.J.: AFIPS,1982), pp.391-392.

TABLE I -COMPARISON OF 4GL CATEGORIES[JENKINS AND SCHUSSEL]

CATEGORY	4GL	PRODUCES APPLICATION	PRODUCES 3GL CODE	PRODUCES PART OF AN APPLICATION	EXAMPLE
APPLICATION GENERATOR [JENKINS]	YES	YES	NO	NO	FOCUS
CODE GENERATOR [JENKINS]	YES	NO	YES	NO	GAMMA
PRODUCTIVITY ENHANCEMENT TOOLS [JENKINS]	NO	NO	NO	YES	COBOL REPORT WRITER
INTERPRETIVE PROGRAMMING LANGUAGES [SCHUSSEL]	YES	YES	NO	NO	FOCUS
FUNCTION GENERATORS [SCHUSSEL]	YES	YES	NO	NO	IDEAL
COMPILED SYSTEM GENERATORS [SCHUSSEL]	YES	NO	YES	NO	GAMMA

1) data base management systems designed for mainframes that include a 4GL for report generation, for query, and for prototyping business computer applications; 2) relational data base management programs designed, initially, for personal computers with integrated spreadsheets and other functions, including a 4GL for applications development; and 3) 4GL's designed originally as application development tools.

To be considered a 4GL, a language must firstly, be tied to, or incorporate, a database management system, which includes backup, recovery, and security features, as well as a data dictionary. It must use a nonprocedural language which has the following features: associative referencing, aggregate operators, elimination of arbitrary sequencing, and pattern directed structures. It must have the ability to handle some complex logic with procedural code and incorporate an interactive query facility and report generator. It should incorporate a screen formatter and offer a productivity improvement over COBOL. Usually this is achieved via a reduction in the number of lines of code, and a reduction in the number of hours of programming effort. Lastly, it must make intelligent default assumptions concerning what the user needs.

Some existing products have only some of the above characteristics. They can be classified as productivity enhancement tools, not 4GL's. Simple report writers, query languages, graphics packages, databases and screen generators do not qualify. Rather, all of these features must be combined in an integrated package to produce a fourth generation language. Figure 1 illustrates a good 4GL

environment.[10]

Given the above characteristics the term "fourth generation language" still remains generic rather than specific. Products which have all of the above characteristics do not necessarily have similar design or similar syntax. This is the result of the multitude of vendors in the 4GL field.

For the purpose of this thesis, third generation languages will be defined as languages which firstly, obtain their data from files rather than databases, and are "procedure-oriented". In other words, the programmer must specify "how" rather than what he wants to accomplish. In addition they may involve data typing, and are domain independent. Examples of 3GL's include PASCAL, BASIC, COBOL, FORTRAN, and Pl/1.

It is obvious that 4GL's have different characteristics than languages such as COBOL, and PL/1. As a result, learning a 4GL might be easier, or more difficult than learning a third generation language.

2.2 CLAIMS MADE ABOUT FOURTH GENERATION LANGUAGES

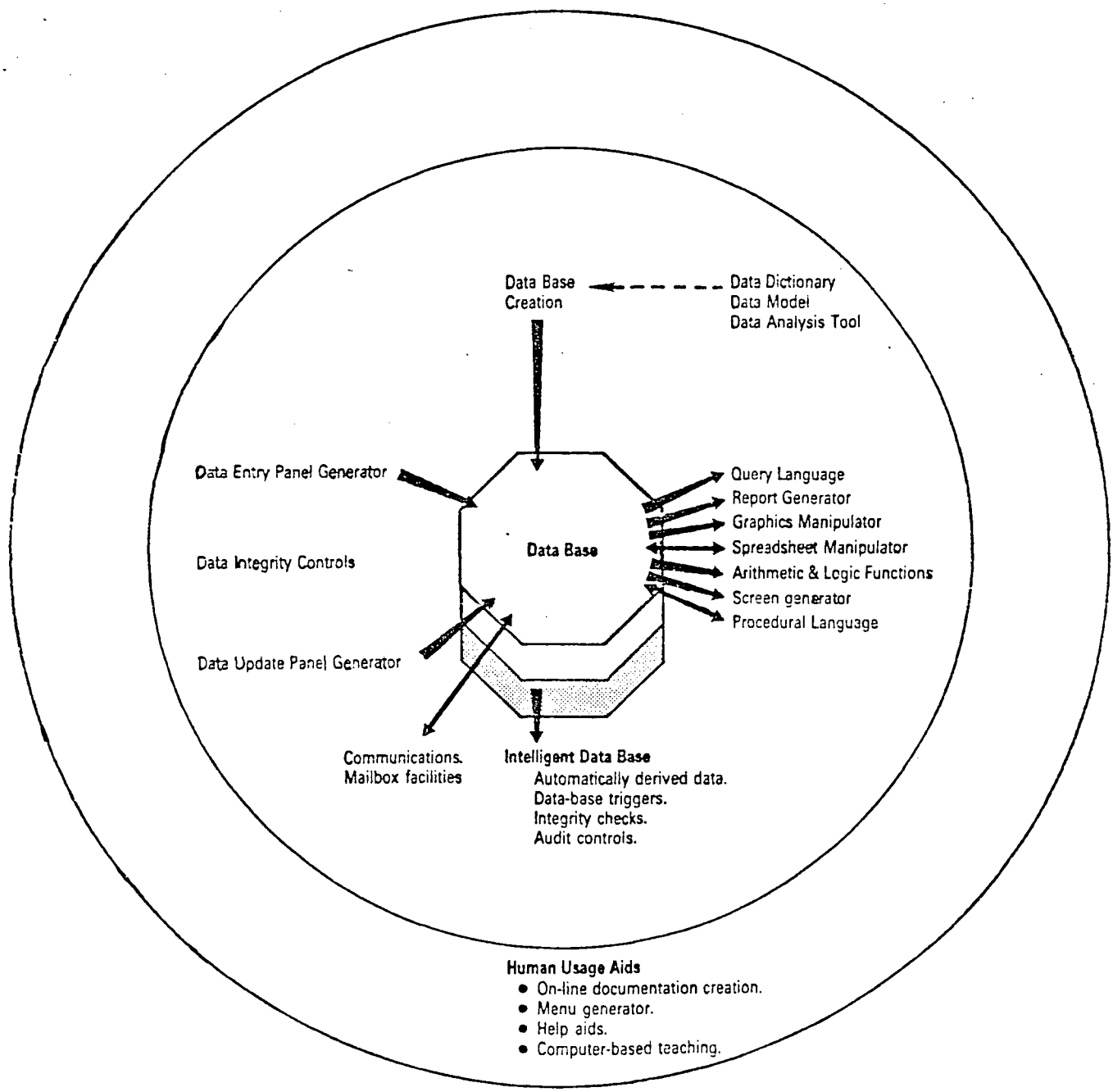
Authors writing on the subject of fourth generation languages have not only failed to better define fourth generation languages, but have also tended to make extravagant claims about the languages.

Generally, these claims have not been backed up by empirical evidence. The following quotes are examples of some of the claims made concerning the capabilities of fourth generation languages. Read and Harmon state: "With 4GL's, programming productivity gains of

---

[10]This illustration is taken from James Martin, Fourth Generation Languages (Lancaster: Savant Institute,1983), p.203.

FIGURE 1 - MARTIN'S MODEL OF A FOURTH GENERATION ENVIRONMENT



1,000% to 2,000% over 3GL's are routinely achieved by personnel with no prior programming experience... About 75% of all programming can be done by end users with only two days of training, specialist programmers, however, will still handle complex applications." [11]Other quotes are similar, "System development with FOCUS takes about one fourth of the time... Programmers multiplied their output by 5 to 10 times by using Ads/Online."[12] James Martin, in his book Application Development without Programmers discusses the 4GL NOMAD. He states: "Beginners who have never programmed find it very easy to achieve results of value by using the nonprocedural statements... It is easy to understand and modify what another person has written in NOMAD... Most end users employ only a small subset of the language. They can be taught to achieve powerful results in a few hours." [13]

The literature also indicates that 4GL's offer a more responsive tool for prototyping because of their interpretive nature and their nonprocedural code. They produce shorter programs because they do not include some of the control statements found in 3GL's, those dealing with input/output format, loop control, handling error conditions, and memory allocation.

Many successful implementations of systems, designed with fourth generation languages, have been reported in the literature. The most famous case occurred at the Santa Fe Railway Co., where a railway reporting system was completely rebuilt by nonprogrammers in a few weeks.[Ibid, p.175]

---

[11]Nigel S. Read and Douglas L. Harmon, "Readers' Forum: Language Barrier to Productivity", Datamation, 29, No.2(1983), pp.209-210.
[12]David Kull, "Nonprocedural Languages: Bringing up the Fourth Generation", Computer Decisions, 15, No.13(1983), pp.156-162.
[13]Martin, Application Development Without Programmers, pp.206-208.

Some writers go so far as to state that neither programming experience, nor technical training are prerequisites for the use of 4GL programming techniques.[14] On the other hand, some writers offer caveats to the claims made by others. Dr Tom Purcell, director of IS with Borg Warner Chemical Corp., states, "As easy as MANTIS (4GL) is to use , and as much as it increases programmer productivity, it's still an order of magnitude too difficult for the average user. Until they make it easier to use, they can't have a programmerless DP department." [15] Wilco tries to dispel some of the myths of 4GL's. She argues that 4GL's are not that easy to learn because of their fairly rigid syntax rules. She also argues that a 4GL cannot be both flexible and easy to use. The more functions that are built in to make it easy the more the user is restricted to the software designer's preconception about what the application system will look like. She concludes that there is still a need for professional programmers.[16]

Read and Harmon also try to dispel some of the myths of 4GL's: "The glossy brochures and magazine ads touting 4GL's often claim nonprogrammers can produce their own reports with little training. On the whole this is true, but what is not explained is that such claims apply to report generation for single applications using single databases employing only a minor part of the full power of the 4GL. With current 4GL's programming complexity rises exponentially with product complexity, and to be functional at the upper levels requires

---

[14] Nigel S. Read and Douglas L. Harmon, "Assuring MIS Success", Datamation, 27, No.2(1981), p.109.
[15] Quoted in an article by Micheal Tyler, "Cincom Shifts Gears", Datamation, 29, No.6(1983), p.65.
[16] Elaine Wilco, "System Development Without Programming", Computer Data, 9, No.2(1984), p.19.

a considerable amount of knowledge and experience."[Read and Harmon, "Assuring MIS Success", pp 118-119]

The literature also indicates that fourth generation languages have a number of weaknesses. For example, 4GL's are resource "hogs" and can use up to 50% more CPU time than 3GL's. Good database design is important in order to minimize the use of resources. The computer using a 4GL must have virtual memory and high speed I/O handling.[Ibid, p.116] Fourth generation languages are not suitable for number crunching operations [Ibid, p.116]. They are weak at character manipulation [Ibid, p.116]. They are not suited for an environment with a high number of transactions per hour - over 30,000 [Cu-Uy-Gam, p.9]. Fourth generation languages also lack language standards. Different vendors offer completely different 4GL's, and even the same 4GL may not be compatible over all hardware. The ease of documentation and maintenance is questionable.[17] There may be a need to fit the software to particular applications. No one type of 4GL is appropriate for all situations. Resistance of existing data processing staff is a problem. [Martin, Application Development Without Programmers, pp.45-47.] Finally, 4GL procedural code is harder to read than COBOL code. When used, the procedural code of a 4GL decreases the nonprocedural benefits of a 4GL dramatically. From his experience, Johnson suggests a limit of 300 statements for 4GL programs. [18] There are exceptions, the EDP Analyzer reports that with the use of Burrough's LINC 4GL, larger programs benefitted more from

---

[17]Paul C. Tinnirelo, "Software Maintenance with Fourth Generation Languages", Proceedings of the National Computer Conference 1984 (Montvale,N.J.: AFIPS,1984), pp.251-257.

[18]James R. Johnson, "A Prototypical Success Story", Datamation, 29, No.11(1983), p.256.

the use of a 4GL than the shorter programs.[19] Considering the weaknesses it is obvious that an Information Systems manager must study his environment carefully before adopting a fourth generation language. This is not to say that a fourth generation language cannot be extremely valuable when used correctly.

The above claims and counter-claims serve only to confuse readers of the strengths and weaknesses of fourth generation languages. From the reports it appears that end users could learn a small portion of a 4GL in a few days and use this knowledge for querying and report generation. But, in order to program large production applications, more knowledge is necessary. It appears that end users, using a 4GL, still cannot develop a large production system. The EDP Analyzer Special Report on 4GL suggests that the applications best suited to a 4GL are those subject to rapid changes, or where the need for ad hoc reporting is high, as in personnel or budgeting.[Ibid, p.15]

Some research has investigated the improvements in programming productivity brought about by using the 4GL FOCUS. Harel and McLean's work lends support to some of the claims of increased productivity although they studied only relatively small systems.[20] Still one question remained unanswered - Could novices learn 4GL's as well as experienced programmers? Read and Harmon offer their opinion: "Since programming techniques with a 4GL are so different from those of earlier generation programming languages, everyone has to start from

---

[19]EDP Analyzer, Special Report: Fourth Generation Languages and Prototyping (Vista,Ca.: Canning Pub.,1984), p.29

[20]Elie C. Harel and Ephrain R. McLean, "The Effects of Using a Nonprocedural Computer Language on Programmer Productivity", MIS Quarterly, 9, No.2(1985), pp.109-119.

square one, which opens up a large new pool of programming talent ... These programming techniques have to be learned from scratch, because there is almost no similarity between programming in COBOL and programming in a 4GL. In fact, for a variety of reasons, a knowledge of COBOL may be a hindrance." [Read and Harmon, "Assuring MIS Success", p.120]. Schleuter in an extract from his book User Designed Computing , quoted in Nicoll-Griffith, adds: " It is a significant fact that the more sophisticated and experienced a DP person is in conventional methodologies, the less likely it is that such a person will be comfortable or even effective when trained to do report processing application design." [21]

While consulting, Martin has encountered the same phenomena. He states:

> "New graduates often learn and become skilled with the new techniques faster than many established programmers. This phenomenon has been observed and measured with many application generators and 4GL's. IBM uses ADF extensively for its own internal development. It has measured the performance of many ADF users and discovered that new graduates do much better on average, than experienced programmers. National CSS staff sometimes refer to the NOMAD programs written by old COBOL programmers as "NOBOL" programs. The COBOL programmers, thinking in COBOL-like terms, fail to use the powerful but different constructs in the NOMAD language".[Martin, Fourth Generation Languages, p.64]

Intuitively, it is reasonable to assume that experience in one programming language would help in learning another, especially when learning and using the procedural aspects of a 4GL. But the above anecdotal evidence points in the opposite direction.

---

[21] Mike Nicoll-Griffith, rev of User-Designed Computing, by Louis Schleuter Jr., MAPPER was the First User Command Language (Montreal: Canadian Pacific Consulting,1983), p.4.

3. RESEARCH RELEVANT TO FOURTH GENERATION LANGUAGES

Because programming is a complex, but poorly defined task, researchers have experienced many problems in conducting experiments with programmers. Most of these problems have surfaced in experiments conducted with third generation languages. The lessons learned from these experiments are also relevant to researchers studying fourth generation languages.

Brooks and Shneiderman discuss the problems caused by the large variation in programmer performance. [22], [23] Brooks states that because the ratio of programmer performance can vary from four-to-one, to twenty-five to one, it is difficult to assemble a group of programmers with equivalent skills. This kind of confound could easily invalidate the results of an experiment. Brooks proposes a large sample as a partial solution. Both Shneiderman and Brooks suggest using within subject tests for experiments where multiple levels are involved.

It is also important that the subjects of the experiment be representative of the population to whom we wish to apply our findings. Brooks suggests that subjects be relatively uniform with regard to their characteristics and abilities at pre-experimentation, in order to avoid introducing confounds. Shneiderman proposes that a lot of data be collected on the subjects. Examining this data for correlations with the dependent variable will help the researcher determine if confounds exist. For example, for each subject, the job

---

[22]Ruven Brooks, "Studying Programming Behavior Experimentally: The Problems of Proper Methodology", Communications of the ACM, 23(1980), p.209.

[23]Ben Shneiderman, "Improving the Human Factors Aspect of Database Interactions", ACM Transactions on Database Systems, 3(1978), pp.423-425.

experience, number of courses taken, of programming courses taken, of languages known, of years programming and of months with each language should be collected. Reisner suggests that an aptitude measure be developed to ensure that the groups are equal. She also lists questions to consider when assessing query languages. Did the subjects have the same kind of background as the people who are expected to use the query language? Were they of the same educational level, the same intelligence level? Was their level of motivation the same as that of the intended users? If some subjects were called "programmers" or "more advanced" how were these classes defined? [24] Prior to the experiment, researchers should make every effort to ensure that subjects are, in all respects, as equal in ability as possible.

Brooks also discusses the kind of programs that should be used in an experiment testing comprehension. Programs which are too easy would result in consistently high scores. Thus, poor variance in the results would be produced, making it difficult to reach any conclusions from the data. Secondly, programs should be representative of real world applications. Brooks suggests 50 to 100 lines of code. Longer programs would be more representative but they would be very hard to administer in a lab experiment.

3.1 MEASURES OF EASE-OF-LEARNING

Shneiderman breaks programming into five tasks: learning, composition, comprehension, debugging, and modification. [Shneiderman, p. 419] As we will see, indications of how well a person has learned programming

---

[24] Phyllis Reisner, "Human Factors Studies of Database Query Languages: A Survey and Assessment", ACM Computing Surveys, 13, No.1(1981), pp.27-28.

has often been measured by requiring the subject to perform any of the other four tasks. For instance, if a novice has thoroughly learned a 3GL such as COBOL, he will be able to write a COBOL program, and understand and modify an existing program. This principle is used extensively in testing university computer science students.

To measure ease-of-learning Brooks,[25] Reisner,[26] and Shneiderman[27] suggest some alternatives: requiring a modification to the program, location of a bug, response to a set of multiple choice questions, a subjective estimation of the clarity of a program, or a hand tracing of the execution sequence. The following questions could be asked for hand tracing: the value of a variable at a specific point in the program, the sequence of values assumed by a variable, the number of times a particular statement is executed, the sequence of statements executed, the output of a program, a brief description of the function of the program, and the impact of an alteration. Some of these tests are not valid for a 4GL because of its nonprocedural nature. The execution of a 4GL is not always strictly sequential.

The literature indicates that researchers have had problems with some of the measures of understanding. Subjective answers by participants of experiments have not proven to be reliable. Open-ended questions can be difficult to score. Weissman tried to develop good measures of understanding while doing experiments on the affects of certain variables (e.g. comments,structure) on the complexity of a program. [28] He concluded that although hand simulation is not a valid

---

[25]Brooks, pp.211-213.
[26] Phyllis Reisner, pp.17-19.
[27]Ben Shneiderman, "Exploratory Experiments in Programmer Behavior", International Journal of Computer and Information Sciences, 5, No.2(1976), pp.125-126.
[28]Larry Weissman, "Psychological Complexity of Computer Programs: An

measure of understanding, it is an important factor which can contribute to one's understanding of a program. Quiz scores tended to go up after hand simulation. He also found that fill-in-the-blank questions were inadequate and decided to use open-ended quiz questions instead.

Shneiderman was not satisfied with existent measures, so he developed a new measure of comprehension: memorization/recall. [29]He presented programmers with scrambled and unscrambled FORTRAN programs and they were asked to memorize them in a given time. They were then asked to reproduce the programs. He showed that more experienced programmers could better reproduce the unscrambled programs because of their chunking ability. The results for scrambled programs were not significant. He hypothesized that memorization and comprehension were correlated. Unfortunately, the results of other studies force us to question the validity of this measure. In Vessey's study, this measure did not correlate well with other measures of programming skill, and did not correctly predict programmer performance.[30]

Composition skills can be measured by asking the subject to write a program, or part of a program. The problem here is that writing a complete program can be time-consuming, and writing a short program can invalidate the external validity of the research. In fourth generation languages, programs are usually short, so a short program would still be valid.

---

[28](cont d) Experimental Methodology", SIGPLAN Notices, 9, No.6(1974), pp.30-34.

[29]Ben Shneiderman, "Measuring Computer program Quality and Comprehension", International Journal of Man-Machine Studies, 9(1977), pp.465-478.

[30]Iris Vessey, An Investigation of the Psychological Processes Underlying the Debugging of Computer Programs, Unpublished Doctoral Dissertation, University of Queensland, Australia (1984), pp.218-220.

When measuring ease-of-use of a query language, Reisner (Reisner's use of "ease-of-use" here is equivalent to my 'ease-of-learning'), suggests some tasks that can be used as measures. The tasks are query writing, query reading (translate meaning into English), query interpretation (what will it do given the data), question comprehension, memorization, and problem solving (given a problem, what queries will solve it).[Reisner, pp16-17] She also lists some kind of tests used to measure ease-of-use: final exams of learning, immediate comprehension (tests while teaching), reviews, retention (how well can the language be used after a long period of time), and relearning. To date, the bulk of the research done in query languages, has used the task of query writing, and final exams, immediate comprehension and retention tests as measures of ease-of-use. Shneiderman argues against using time as a measure of quality because those who finish first are not necessarily the best.[31] He supports setting a fixed time length for task performance because it focuses attention on correctness and quality.

Before we can measure subject comprehension the subjects have to be taught the language. To date, most researchers have used the traditional classroom method. Reisner[32] justified the method and stated that "classroom teaching is relatively quick to implement and known to be effective, and because it provides opportunity for on the spot feedback between the teacher and the student.' But she admitted that computer instruction would have been more reproducible. If more

---

[31]Shneiderman, "Improving the Human Factors Aspect of Database Interactions", p.426.

[32]Phyllis Reisner, "Human Factors Evaluation of Two Data Base Query Languges - Square and Sequel", Proceedings of the National Computer Conference 1975(Montvale, N.J.: AFIPS,1975), p. 451.

than one class is taught, or if multiple instructors teach different classes, the equivalence of teaching between classes is hard to establish. A more equal method would be to teach, employing a training manual. This method has the added advantage of being most representative of how workers learn on the job.

3.2 RESEARCH ON FOURTH GENERATION LANGUAGES

To date, research of fourth generation languages has concentrated on the productivity advantages of 4GL's over third generation languages. As yet, no research has examined the ability of experienced third generation language programmers, and novices, to learn fourth generation languages. The two studies summarized below are the only pieces of research conducted in the area of fourth generation languages up to this point.[33]

Munnecke conducted a descriptive study of a fourth generation language.[34] He compared the fourth generation language ,MUMPS, used at the Massachusetts General Hospital, with COBOL, on a strictly linguistic basis. He believed that a computer language should support users, linguistically, on their own terms, to adapt to their needs, and to be as forgiving and friendly as possible. He compared the access methods of COBOL/IMS with MUMPS. COBOL/IMS has 12 different complicated access methods while MUMPS has only one simple access method. Also, COBOL/IMS database pointers involve complicated physical references while MUMPS navigates more logically. The amount of

---

[33] There has been a lot written in the practitioner literature, but, as I mentioned earlier, it cannot be considered "research".

[34] Thomas Munnecke, "A Linguistic Comparison of MUMPS and COBOL", Proceedings of the National Computer Conference 1980, (Montvale,N.J.: AFIPS,1980), pp.723-729.

documentation that supports each system is also compared. Over 1700 pages must be read for COBOL/IMS. MUMPS documentation is much shorter. Munnecke concludes that the 4GL is better suited to meeting the needs of users, within the range of applications it can handle.

Recently, Harel and McLean studied the effects of using a nonprocedural language on programmer productivity.[Harel and McLean, pp.109-119]. A field experiment was conducted in order to compare COBOL with the 4GL FOCUS, in terms of programmer productivity and program efficiency. Beginners (people who had programmed less than 20 programs in that language) and experts (more than 20 programs) were asked to program report generation applications (simple and complex). In every case, the 4GL FOCUS was found to be faster but less efficient in terms of machine resources used. Programmers, with little experience, did significantly better with FOCUS than COBOL, while the difference for experts, was not as great. This suggests FOCUS might be a good end user language, but this conclusion is weakened by the fact that the less experienced programmers were far from being novices, and, in fact, all the programmers were professionals. Also the "complex" applications were not really complex. But neither were they overly simple. The complex task took approximately three days to program in COBOL. In longer programs, COBOL might have had an advantage over FOCUS.

The results of the study imply that FOCUS would be a good end user language because novices can learn it quickly. On the other hand, the experiment does not deal with the issue of whether novices or experienced programmers learn 4GL's more easily. Programmers need to be tested more directly to see if novices can learn and use a 4GL

as easily as an experienced 3GL programmer.

3.3 QUERY LANGUAGE RESEARCH

Some comparison of novice versus experienced programmer learning has been conducted with query languages. Query languages are similar to 4GL's because they are nonprocedural, and because they are often incorporated into 4GL's.

Reisner et al. compared two data base query languages, SQUARE and SEQUEL, using both novices (no programming experience) and programmers (had taken one programming course).[35] Their main finding was that SEQUEL was easier to learn than SQUARE, but she also had other interesting results. Programmers were able to learn the new nonprocedural languages somewhat faster than nonprogrammers (12 hours versus 14 hours of class time). The difference in scores between programmers and nonprogrammers on the quizzes were significant ($p < .01$). Programmers scored higher than nonprogrammers. On the one hand, since query languages are similar to 4GL's, we might expect the same results for subjects learning a 4GL. On the other hand, the learning time comparison must be viewed with skepticism. Reisner explains that the pace of classes was determined by the slower learners. Therefore, a few slow learners in one of the classes could seriously affect the learning time of the whole class, throwing off the validity of any comparisons between classes.

In testing the students, Reisner et al. used five review quizzes during the classes, a final exam at the end, and a memory test one

---

[35] Phyllis Reisner, Donald D. Chamberlain, and Raymond F. Boyce, "Human Factors Evaluation of Two Database Query Languages -Square and Sequel", Proceedings of the National Computer Conference 1975 (Montvale,N.J.: AFIPS,1975), pp.447-452.

week later. With the exception of the memory test, students were allowed to use reference materials.

Based on their results, Reisner et al. argue that query languages can be learned in layers.[36] Given the basic subset of a query language, an inexperienced programmer can learn to use the language very quickly with few errors. Because this subset is so simple, experienced programmers may not have an advantage in learning when only the basics are considered. Their advantage may only become apparent when the more complex procedural aspects are taught. The same results might be expected in this research when scores on the simple task are compared between novices and experienced programmers.

In a similar study, Welty and Stemple compared the nonprocedural query language SEQUEL with the procedural query language TABLET.[37] They believed there was a point of complexity in languages beyond which a procedural language is easier than the nonprocedural. They defined a metric for procedurality and went on to show that TABLET was a better language for complex queries. Again, programmers outperformed nonprogrammers in learning the language (significant at the .05 level). They also had a higher retention score one week later. The scoring, teaching and subject classifications were basically the same as Reisner's. Again, this seems to imply that experienced programmers have an advantage in learning a 4GL, especially the more procedural aspects used for complex applications. But the study also reveals that, when the quiz scores for the

---

[36] Phyllis Reisner, "Use of Psychological Experimentation as an Aid to Development of a Query Language", IEEE Transactions on Software Engineering, 3(1977), p.222.

[37] Charles Welty and David W. Stemple, "Human Factors Comparison of a Procedural and a Non-Procedural Query Language", ACM Transactions on Database Systems, 6(1981), pp.626-649.

experienced programmers are compared, SEQUEL scores are lower on average than the SQUARE scores. This is not the case for inexperienced (novice) programmers. Again, this may mean that programmers do not have as large an advantage when learning a nonprocedural language. Still, the results indicate that experienced programmers can outperform novices when learning query languages.

3.4 THIRD GENERATION LANGUAGE RESEARCH

Chrysler, in his study of what affects the productivity of programmers, found that experience does have significant impact. [38] He measured the following variables: the number of months of programming experience, the number of months of experience using COBOL, the number of months experience using the specific COBOL compiler, the number of months experience in programming business applications, and the number of months experience programming for the current employer. Programmers developed code in the COBOL language. All variables were found to be significantly correlated to the productivity of programmers. Even the number of months programming experience (not necessarily all in COBOL) was significant. This indicates that experience with one third generation language, improves performance in another 3GL.

In another COBOL study, Gordon et. al. showed that programmers with at least three years experience in COBOL, outperformed students who had just learned COBOL.[39] The professionals finished their programs with less errors and fewer runs. These studies indicate that experience with one 3GL assists when using another 3GL, and the more

---

[38] Earl Chrysler, "Some Basic Determinants of Computer Programming Productivity", Communications of the ACM, 21(1978), pp.472-483.

[39] J.D. Gordon, A. Salvadori, and C.K. Capstick, "An Empirical Study of COBOL Programmers", INFOR, 15(1977), pp.229-241.

experienced the programmer the better. But these results may not apply in a 4GL environment.

Kennedy studied the learning curves of naive users of a new system. He showed that the anxiety and fear of naive users can have an effect on their ability to learn.[40]

DuBoulay and O'Shea, report a study by Mayer comparing different program structures.[41] The GOTO, IF THEN and a nonprocedural construct were compared for comprehension. The novices were not asked to write programs but to interpret and answer questions about a sporting competition, expressed in the various program-like forms. Results indicated that the nonprocedural construct was the most comprehensible. The harder the question was, the greater the superiority of the nonprocedural representation. This study is weak, in the sense that real programs and programmers are not involved. Yet, it again indicates that nonprocedural structures are easier for novices to comprehend, which has some implications for learning a 4GL. This indicates that novices are at less of a disadvantage, as compared to experienced programmers, when learning a nonprocedural language, than when learning a procedural language.

Some research has been conducted on debugging programs. Youngs used thirty novices and twelve professional programmers in his study of debugging. [42] Novices were university students taking their first programming course and professionals held or had held professional

---

[40]T.C.S. Kennedy, "Some Behavioural Factors Affecting the Training of Naive Users of an Interactive Computer System", International Journal of Man-Machine Studies, 7(1975), pp.817-834.

[41]B. DuBoulay and T. O'Shea, "Teaching Novices Programming", Human Interactions with Computers, ed H.T. Smith and T.R.G. Green (New York: Academic Press,1980), pp.159-162.

[42]Edward A. Youngs, "Human Errors in Programming", International Journal of Man-Machine Studies, 6(1974), pp.361-376.

programming jobs. Youngs compared the number of errors committed by the two groups. Experienced programmers committed fewer errors and corrected their programs more quickly. Youngs also discovered that novices made many more syntax and semantic errors than the professionals but the same number of logic errors. He pointed out especially troublesome areas for novices, like looping and input /output formatting. If these could be eliminated (as in 4GL's), novices could compete more evenly with professional programmers. This could explain why novices can learn 4GL's as well as experienced 3GL programmers.

A debugging study conducted by Vessey concluded just the opposite.[Vessey, pp.206-222] She showed that more experienced programmers do not necessarily debug better than less experienced programmers. Managers classified programmers as experts or novices on the basis of the number of years experience they had accumulated. Vessey concluded that this classification was not a good predictor of debugging performance. Even though Vessey was dealing with debugging and not programming, her study indicated that the number of years of programming alone cannot predict how proficient a person will be at programming.

In conclusion, it is obvious that there is a need for research in the area of fourth generation languages. The question of whether novices can prepare their own applications with fourth generation languages, and whether or not 3GL programmers can transfer their skills into a 4GL environment, have not been addressed. The one thing that we can conclude from the research done to date is that experienced programmers have always outperformed novices in using

query languages and third generation languages.

4. THEORY

How well a person can learn a fourth generation language, after using a third generation language, is a question of transfer of training. Most of the theory relevant to learning comes from psychology. These theories have yet to be applied to programming studies. The most important theories involve the concepts of positive and negative transfers in learning. Garry and Kingsley explain the theory as follows:

> "When training in one situation or one form of activity affects one's ability in another type of activity or one's performance in different situations we have what is commonly understood as transfer of training. An attempt to operate a tractor or a truck based upon one's knowledge of operating an automobile requires transfer of training in order to succeed in the task. In countless ways we use the results of past learning to meet the demands of new situations. In many ways the results of past learning interfere with new learning, for instance, the difficulty we experience in correctly pronouncing a foreign language because of our habitual manner of pronouncing sounds."[43]

If prior experience facilitates learning in a new situation, we say that positive transfer has occured. Osgood showed that the key factor is the similarity of stimuli in different situations, when the same behaviors are required. [44]

When prior learning interferes with learning in a new situation we say that negative transfer has occurred. Typically negative transfer occurs when persons are required to learn new responses to stimuli to which other responses have, previously, been learned. An example is, learning to drive a car with manual transmission after having learned on a car with automatic transmission.

---

[43]Ralph Garry and Howard L. Kingsley, The Nature and Conditions of Learning (Englewood Cliffs,N.J.: Prentice Hall,1970), p.512.
[44]C.E. Osgood, Method and Theory in Experimental Psychology (New York: Oxford,1953), p.495-548.

The question is - which one of these cases applies to an experienced third generation language programmer attempting to learn a fourth generation language? When fourth generation languages are examined, we notice that their commands accomplish more than third generation commands. In other words, one fourth generation language command is equivalent to a number of third generation language commands. For example, the COUNT command in FOCUS is equivalent to a third generation language DO loop:

```
I=0
WHILE NOT END-OF-FILE DO
   READ RECORD
   I = I + 1
END WHILE
COUNT = I
```

In some cases a 4GL command is directly equivalent to a 3GL command. For instance, the TYPE command in FOCUS has a similar function to the WRITELN command in PASCAL. Because programmers use the same algorithm to solve the problem no matter what the language, 3GL programmers should be able to translate some of their skills to a 4GL environment. But, the different syntax, nonprocedurality, and conciseness of fourth generation languages could make the transfer of skills very difficult.

We gain some insight into which of the above two possibilities is more likely to be true by examining two theories of learning transfer. Garry et al. describe the two theories.[Garry and Kingsley,

pp.513-531] The theories have opposing views of the conditions which make transfer of training possible.

The theory of transfer by similarity states that the more two functions have in common, the more likely it is that training in the first will tend to improve the second. The commonality of the two situations is measured by the constituents, or components of the situation. The mere presence of common components does not assure positive transfer; under some conditions of training, they produce negative transfer. The amount of transfer, due to identical features of two functions, varies with the locus of identity or the phases of the functions in which identity occurs. Identity, in the response phase, is conducive to far more positive transfer than identity in the stimulus factors. Thus, it is easier to learn to respond to a new situation in an old way, than it is to develop a new method of response to an old situation, for, in the latter case, the interference from previously formed habits is greater. Since, in our case, we have a new method of response (4GL) rather than a new situation, transfer is more difficult.

An opposing view is developed in the theory of transfer through relationships. This theory argues that positive transfer is due not only to similarity of content, but also to the similarities in the patterns of relationships. It is often claimed that competitive athletics contribute greatly to the successful performance of an American soldier in combat. According to the relationship theory, transfer occurs because both activities involve coordinated teamwork of individuals performing related operations, not because of the similarity of the required abilities (speed, strength, agility). The

strategy involved in out thinking and out manoeuvering an opponent would be more important than the specific tactics employed. If this were the case in programming, 3GL programmers should be able to transfer their training to 4GL's because both tasks involve individuals solving business problems using data processing skills. In this case, it is not the similarity of syntax which is an important condition of transfer, but rather it is the overall data processing strategies and knowledge that make transfer possible.

The majority of theories of programming developed so far support transfer through relationship. Shneiderman's work on comprehension, hypothesizes that experienced programmers can outperform novices because they have better chunking ability than novices.[45] Experienced programmers have a better understanding of the semantics and logic of a program so they can view it at a higher level than novices (at the problem level rather than at the syntax level). In other words, the most important skills acquired by a programmer are the semantic and logic skills, not familiarity with the syntax. It would be reasonable to assume that once these skills are acquired, programmers could use these skills in many language environments. Shneiderman explains his model as follows. The programmer first conceives the problem in general terms such as general programming strategies. He refers to these general plans as "internal semantics". He suggests that this internal representation progresses from a very general outline to a more specific plan, to a specific generation of code focusing on minute details. Shneiderman's "funneling" view of problem solving (going from general to specific) was first introduced as a result of

---

[45]Ben Shneiderman, Software Psychology: Human Factors in Computer and Information Systems (Toronto: Little, Brown and Co.,1984), pp.46-53.

Duncker's experiment based on asking subjects to solve complex problems aloud.[46] Once the programmer has worked out the internal semantics the construction of a program is a relatively straightforward task. The programmer draws on his knowledge of semantic structures and syntax to write the code. The program may be composed in any familiar programming language.

Other authors have developed theories along the same lines. Chase and Simon conducted similar work with chess players. [47] They showed that experienced chess players could memorize chessboard positions more easily than less experienced players. Simon used the "chunking" hypothesis to explain this phenomena. Higher level chess players do not memorize the position of each piece, but rather memorize meaningful "chunks" of pieces.

Mayer hypothesizes that experienced programmers have an advantage over novices because they have "anchoring ideas" in long term memory, which they use when learning. He explains: "In the course of meaningful learning the learner must come into contact with the new material, then must search long term memory... for anchoring ideas and then must transfer these ideas to short term memory so that they can be combined with new incoming information."[48] Novices are at a disadvantage because they have not encountered similar syntax, and because they do not have built in algorithms. Some programmers can immediately identify the purpose of loops, subprograms and other structures because they have seen similar constructs before.

---

[46] K. Duncker, "On Problem Solving", Psychological Monographs, 58(1945), p.270.
[47] William G. Chase and Herbert A. Simon, "Perceptions in Chess", Cognitive Psychology, 4, No.1(1973), pp.55-81.
[48] Richard E. Mayer, "The Psychology of How Novices Learn Computer Programming", ACM Computing Surveys, 13, No.1(1981), p.122.

The remaining question to be considered is: Do the differences between fourth generation languages and third generation languages require that the strategies and algorithms used to attack a programming problem also be changed? Shneiderman would base his answer to this question on the semantic similarity of the two languages. He states: "Learning a first language requires development of both semantic concepts and specific syntactic knowledge, while learning a second language involves learning only a new syntax, assuming the same semantic structures are retained." [49] Though 4GL's are very different, they are used to accomplish the same tasks as 3GL's and must therefore be semantically similar. To use 4GL's in complex tasks, programmers must use the same concepts used in 3GL's, specifically, control breaks, file structures, and field formats. An experienced 3GL programmer will have a thorough knowledge of these concepts. This will help him in any 4GL work.

To recap, Shneiderman's and Simon's work indicate that experienced programmers make better programmers than novices because, among other things, of their ability to chunk problems. They emphasize that insight into the semantics and logic of a program is more important than mere syntactic knowledge. Further to this point 4GL's are semantically similar to 3GL's because they are used to accomplish the same goals. Therefore, experienced programmers should be able to transfer their skills to 4GL's. Research in query languages, which are very similar to 4GL's, support this conclusion. Reisner's and Welty's research in query languages has shown that experience in other programming languages does improve performance with query languages.

---

[49] Software Psychology: Human Factors in Computer and Information Systems, p.48.

Accordingly, if experienced programmers and novices are tested on their ability to learn 4GL's, experienced third generation language programmers should record higher mean scores than novices on both simple and complex tests of fourth generation languages. This is empirically tested using the following one sided test:

Ho: Experienced 3GL programmers' scores on 4GL tests will be equal to novices' scores on 4GL tests

Ha: Experienced 3GL programmers scores on 4GL tests will be greater than novices' scores on 4GL tests

The practitioner literature indicates that novices can program in fourth generation languages, but, once the application becomes complex, experienced data processing people are needed. When novices are faced with more complex applications, they lack what Mayer calls "anchoring ideas". They lack the data processing concepts which would help them understand these problems. In 4GL's, as problems become more complex, the 4GL commands used become many times as hard, semantically, than the more simple commands. Shneiderman has hypothesized that knowing the semantics of a language allows programmers to transfer their skills to other languages. Since 3GL's and 4GL's are semantically similar, experienced 3GL programmers should be able to apply their experience to complex 4GL problems. Accordingly, the difference in test scores, between simple and complex 4GL tasks, should be greater for novices than for experienced 3GL programmers. This is empirically tested as follows:

Ho: Difference in simple and complex test scores for experienced programmers is equal to the difference in simple and complex test scores for novices

Ha: Difference in simple and complex test scores for experienced programmers is less than the difference in simple and complex test scores for novices.

Finally, from previous work with programmers, Shneiderman and Reisner have found that other variables such as work experience and prior experience with other programming languages improve programming performance. Accordingly, the third hypothesis to be tested is: the number of query language programs written, number of report writer programs written, and number of 4GL programs written, will be positively correlated to 4GL test scores.

5. METHOD

To test the developed hypotheses, a laboratory experiment was conducted in which the performance of novices, and experienced third generation language programmers, using a fourth generation language was measured across two levels of task complexity.

5.1 PARTICIPANTS

Fifty-seven volunteers, from two different educational institutions, participated in the study. Twenty-four students (mostly MBA students) at the University of British Columbia (UBC) took part in the experiment, as well as thirty-three Computer Systems Diploma (two year program) students from the British Columbia Institute of Technology (BCIT).

These two groups of students were used in order to improve the external validity of the study, and to provide the necessary mix of novice and experienced programmers. The MBA students were chosen for two reasons. Firstly, these students had little programming experience, and, therefore, provided a supply of novices for the experiment. Secondly, as business students, they represented the end-users of the future, people who have little knowledge of computers, considerable knowledge within business areas, and may be doing the programming in the future. The Computer Systems Diploma students were also chosen for two main reasons. They provided a supply of experienced student programmers for the experiment. The second reason again relates to the issue of external validity. It was felt that Computer Systems students were more representative of the type of programmer who would use a fourth generation language.

Specifically, Computer Systems students were preferred over university computer science students because their education stressed business programming, especially the use of COBOL, whereas university computer science programs place more emphasis on scientific computing and theory. Since the experiment specifically tested the use of a fourth generation language, in a business reporting setting, business oriented students were required rather than scientifically oriented students. For this reason, university computer science students were ruled out.

The difficulty in using two distinct groups of subjects was that educational background could become confounded with experience. In order to minimize the chances of other variables affecting the results of the experiment, two steps were taken. The first step involved using the educational institution as a blocking variable in the statistical analysis. This will be explained in more detail in the Design section of this thesis. The second step involved collecting information on other possible confounding variables (e.g. number of query language programs written), and regressing them against scores obtained on the fourth generation language tests. This analysis revealed whether any other variables, besides experience with a third generation language, could explain a person's ability to learn and use a fourth generation language.

It was thought that the advantages of having two representative groups outweighed the mentioned disadvantages. Using only Computer Science or MBA students could have cast doubts on the external validity of the experiment.

5.2 DESIGN

The experimental design used was a randomized block design (See Figure 2). Two factors were studied, experience with third generation languages, and task complexity. Each factor had two levels. In addition, educational institution was used as a blocking variable.

5.2.1 COMPLEXITY FACTOR

One of the two factors was task complexity. After learning some FOCUS fourth generation language reporting commands, subjects were asked to write a test involving either, a simple reporting task, or a complex reporting task. Two levels of task complexity were used in order to test hypothesis number two, concerning the ability of novices to program complex applications. The subjects were randomly assigned to one of the two tasks. The first task tests the subjects' knowledge of a set of approximately seventeen FOCUS commands. These commands form the basic subset of FOCUS reporting commands. They are the easiest to learn, and can be used to generate reports involving printing, summing, counting, and subtotalling. The second task tested the subjects' ability to produce more complex reports involving multiple files, temporary fields, complex summarization and more detailed formatting. The commands needed to accomplish these tasks increased the subjects' required command set by seven. Including the simple commands, this increased the subjects' inventory of commands to twenty-four. Thus, the task became more complex for two reasons. Firstly, the user had to have a larger vocabulary of FOCUS commands, and, secondly, the user had to be able to understand the workings of the more complex commands.

FIGURE 2 - EXPERIMENTAL DESIGN

UBC

3GL PROGRAMMING EXPERIENCE

TASK COMPLEXITY	NOVICE	EXPERIENCED
SIMPLE		
COMPLEX		

BCIT

3GL PROGRAMMING EXPERIENCE

TASK COMPLEXITY	NOVICE	EXPERIENCED
SIMPLE		
COMPLEX		

The boundary between the two categories (simple and more complex) is not extremely sharp. Other than the above reasoning, no precise method of categorization is advanced. Notwithstanding, there is no doubt that the complex test was more difficult because of the additional commands needed. Indeed, the analysis of variance that was performed later, indicated that subjects scored much lower on the complex test. The difference in test scores between simple and complex tests was significant (p value = .0011).

Both the complex and simple commands were explained in the Report Generation Manual. Although there were only seven complex commands, they had to be read more carefully to be understood, and the user needed practice before he or she was able to understand how they worked. The simple commands were more intuitive. In addition, using the more complex command took more time because the user had to consider the many implications (results) of the command. As the commands became more complex the user had to consider the effect of using one command in combination with others. For example, the SUBHEAD command could only be used after consideration of the effects of the SUBTOTAL command.

The two groups of commands used in the tests appear in Table II below, along with the reasoning for placing them in the appropriate category.

TABLE II - CATEGORIZATION OF FOCUS COMMANDS

COMMAND	CATEGORY	REASONING
TABLE FILE	simple	Used to begin reporting sesssions. User has only to type this as well as the filename of the file he wants to use.
PRINT, SUM, COUNT	simple	These are the basic verbs. Meanings are the same as their normal English definition.
HEADING, FOOTING	simple	Prints headings. The user has only to enclose his text in quotes.
IF	simple	Used for record selection. Easy to understand because it has the same meaning that "if" does, used in everyday vocabulary.
BY, ACROSS, OVER	simple	Simple sort display commands. Either sorts the fields horizontally, vertically or prints them over each other.
SUBTOTAL, SUB-TOTAL	simple	Simplest of totalling commands. Can only add fields together at the sort break.
AS	simple	Used to replace headings with another name supplied by the user. User has only to supply the new name in quotes.

NOPRINT	simple	To suppress printing of a field. User specifies the field to be suppressed.
UNDER-LINE, SKIP-LINE PAGE-BREAK	simple	User has only to specify the field he wants this action to affect.
JOIN	complex	User requires understanding of indexes, and field formats. Join involves the use of at least two files. User also needs to know how to refer to the joined file, and how the JOIN is related to DEFINE and TABLE.
DEFINE, COMPUTE	complex	Used to create a new field from those given. Involves other concepts such as concatenation and assigning new field values selectively.
RECAP,SUMMARIZE	complex	Used to produce other than simple subtotals at the control break. User needs an understanding of both COMPUTE and SUBTOTAL combined to use these control break commands.
SUBFOOT, SUBHEAD	complex	Used to print summary at control break. Need an understanding of where control breaks will occur. Also need to know how to print current database field values in the text.

When the user begins to use the complex commands, an understanding of database theory is an advantage. Commands like JOIN

and DEFINE require some understanding of database concepts. The simple commands are more similar to the third generation language commands, than the more complex commands.

5.2.2 EXPERIENCE FACTOR

The experince factor had two levels, novice, and experienced 3GL, and were classified by information provided in the questionnaire. Three independent judges determined whether the subjects were novice programmers or experienced 3GL programmers. One of the judges was an MIS academic, and the other two were MIS professionals. The subjects were classified on a seven point scale according to their prior experience with third generation languages. Experience with other software, such as query languages, or database management systems, was not considered. Experience with other types of software was not considered because the purpose of the classification was to test hypothesis number one, concerning the influence that prior work with third generation programming languages had on the ability to learn a fourth generation language. Experience with these other factors are considered under hypothesis number three. The instructions that were distributed to the judges appear in Appendix 1. The judges ratings appear in Table III. The reliability between judges as measured by the correlations were: .65 for judges one and two, .78 for judges one and three, and .68 for judges two and three. From the results, we can see that judge number two used a slightly different rating scale than judges one and three, but, overall, the ratings were similar. The

TABLE III -JUDGES' RATINGS OF THE SUBJECTS' EXPERIENCE

SUBJECTS	JUDGE1	JUDGE2	JUDGE3	MEAN
1	1	3.0	2	2.00
2	2	4.5	4	3.50
3	1.0	1	1	1.00
4	4.5	5	5	4.83
5	1.0	1	1	1.00
6	4.5	5	4	4.50
7	4.5	7	5	5.50
8	5.0	3	5	4.33
9	4.5	4	4	4.17
10	4.5	3	3	3.5
11	3.5	1	2	2.17
12	4.5	4	4	4.17
13	3.5	1	2	2.17
14	4.5	3	4	3.83
15	4.5	3	6	4.50
16	4.5	3	4	3.83
17	4.5	5	4	4.50
18	3.5	2	3	2.83
19	5.5	5	6	5.50
20	3.0	1	2	2.00
21	4.5	3	3	3.50
22	2.0	1	2	1.67
23	1.0	1	1	1.00
24	5.5	7	5	5.83
25	2.0	1	1	1.33
26	4.5	1	2	2.50
27	5.5	4	4	4.50
28	5.0	2	7	4.67
29	3.5	2	4	3.17
30	1.0	1	1	1.00
31	2.5	3	3	2.83
32	5.0	3	3	3.67
33	5.0	4	4	4.33
34	3.5	3	7	4.50
35	6.0	7	5	6.00
36	1.0	1	1	1.00
37	4.0	2	3	3.00
38	1.0	1	1	1.00
39	5.5	1	5	3.83
40	1.0	1	1	1.00
41	3.5	1	2	2.17
42	5.0	4	3	4.00
43	1.0	1	1	1.00
44	3.5	5	4	4.17
45	6.0	7	5	6.00
46	5.0	2	3	3.33

47	1.0	1	1	1.00
48	2.0	1	2	1.67
49	3.5	1	3	2.50
50	3.5	1	2	2.17
51	4.5	3	3	3.50
52	4.5	2	3	3.17
53	5.5	7	6	6.17
54	4.0	4	4	4.00
55	3.5	2	2	2.50
56	3.5	7	4	4.83
57	3.0	3	3	3.00

mean score of the three judges , for each subject was calculated. These mean scores were plotted on a frequency chart (See figure 3). As can be seen, the double humped distribution, which would indicate obvious novice and expert groups, did not occur.

Since the frequency chart showed no distinct novice and experienced groups, other methods had to be used to form novice and experienced groups. Two methods were used. The first method grouped all subjects rated one, two, or three as novices, and those rated five, six, or seven (though no subjects were rated 7) as experts, and those rated four as intermediates. The intermediate group was not used in hypothesis testing. The second method considered the top forty percent of the subject ratings were experts and the bottom forty percent were novices. The in-between twenty percent were considered intermediates and were dismissed. Figure 4 shows the amount of agreement and disagreement that was found on subject ratings between the two methods. As can be seen, the first method resulted in a smaller expert class than the second method because few subjects were rated as sixes or sevens. Figure 5 shows the breakdown of subjects in each treatment. Method one, the number separation method, resulted in a group of twenty-eight novices and a group of fifteen experts. Method two, the percentage separation method, resulted in a group of twenty-three novices and a group of twenty-two experts. The groups for method two are slightly unbalanced because the novice group included all ratings up to, but not including, three, and the expert group included all ratings of four and above. A twenty-third expert was not chosen to preserve the 3 to 4 range as intermediates.

FIGURE 3 - FREQUENCY CHART OF THE MEANS OF THE JUDGES' RATINGS

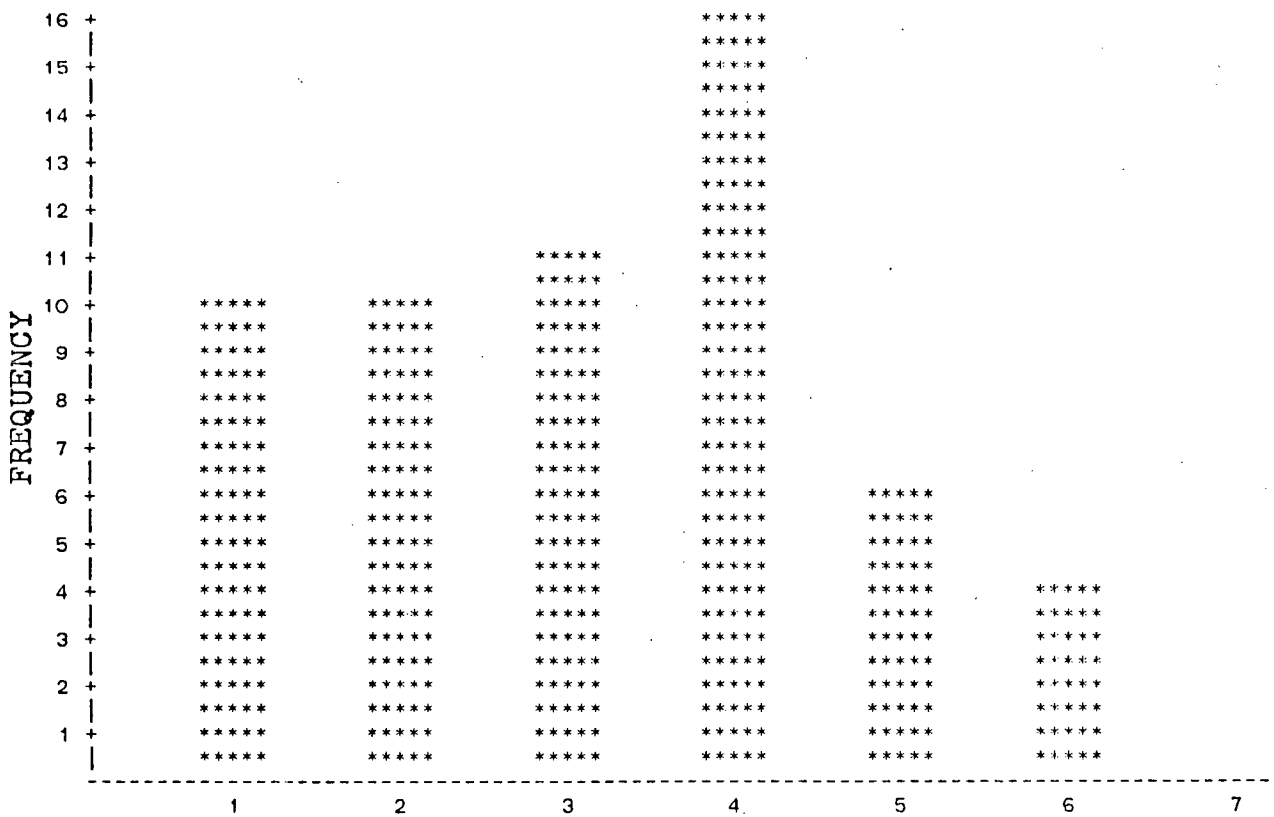


FIGURE 4 - COMPARISON OF SUBJECT SEPERATION METHODS

NUMBER RATING SEPERATION METHOD

PERCENTAGE SEPARATION METHOD	NOVICE	INTERMEDIATE	EXPERT	
NOVICE	23	0	0	23
INTERMEDIATE	5	7	0	12
EXPERT	0	7	15	22
	28	14	15	57

FIGURE 5 - NUMBER OF SUBJECTS IN EACH TREATMENT

TASK DIFFICULTY	PROGRAMMING EXPERIENCE: NOVICE	PROGRAMMING EXPERIENCE: EXPERIENCED
SIMPLE	METHOD 1 = 16 METHOD 2 = 14	METHOD 1 = 6 METHOD 2 = 10
COMPLEX	METHOD 1 = 12 METHOD 2 = 9	METHOD 1 = 9 METHOD 2 = 12

Besides the slightly larger number of experts produced by method two, the two methods of separation resulted in ratings which were almost identical. Statistical tests, testing hypotheses number one and two, were performed, using both separation methods. Identical results were obtained. For this reason, the statistical analyses presented in the rest of the thesis will only show results using the first method of separation. This method is preferred because it uses the judges' ratings directly. It does not force a certain percentage into the expert group as the second method does.

A third variable, a blocking variable, was used to eliminate the differences which might occur between subjects. These differences were due to the location of the testing, or the background of the subjects. The blocking variable (educational institution) removed the variability in the scores caused by location of the experiment. Thus, it cannot be argued that the difference in test scores was due to educational background or experimental setting. In order to use this randomized complete block design, we had to prove that there were no interactions between the factors and the block, and that the blocking variable did not explain a significant amount of variation of the dependent variable. If this blocking variable did explain a significant amount of variation, educational institution would have to be considered a confound. When analysis of variance was performed, educational institution was not found to be a significant factor. Also, educational institution did not have any significant interactions with the other factors. Therefore, educational institution can be used as the blocking variable.

5.2.3 CLUSTER ANALYSIS

"Given a sample of N objects or individuals, each of which is measured on each of p variables, cluster analysis is a classification scheme for grouping the objects into classes."[50] For our purposes, cluster analysis was used to classify the subjects of the experiment into three classes, based upon measures of their programming experience. The three measures used were: the number of third generation programming languages known, the number of third generation programs written, and the amount of programming work experience. The three classes of subjects obtained were novice, intermediate, and expert programmers. This classification was then used as a comparison with the classifications arrived at from the judges' ratings. In other words, cluster analysis was used as a non-subjective tool to lend validity to the judges' ratings.

There are several different methods of cluster analysis. Clusters can be obtained by hierarchical techniques in which a group of subjects is split into smaller groups (or individual subjects are joined into clusters) based on distance measures. The clusters can also be obtained by density techniques which seek regions of high density to form clusters. Clumping techniques, which allow overlap between clusters, are also used, but are not appropriate here because of the need for mutually exclusive groups. Based on his studies, Everitt concludes that the best results seem to be obtained by hierarchical techniques. [Everitt, p.45]

Several hierarchical techniques were used, but the best results come with Ward's method. "Ward proposes that at any stage of an

---

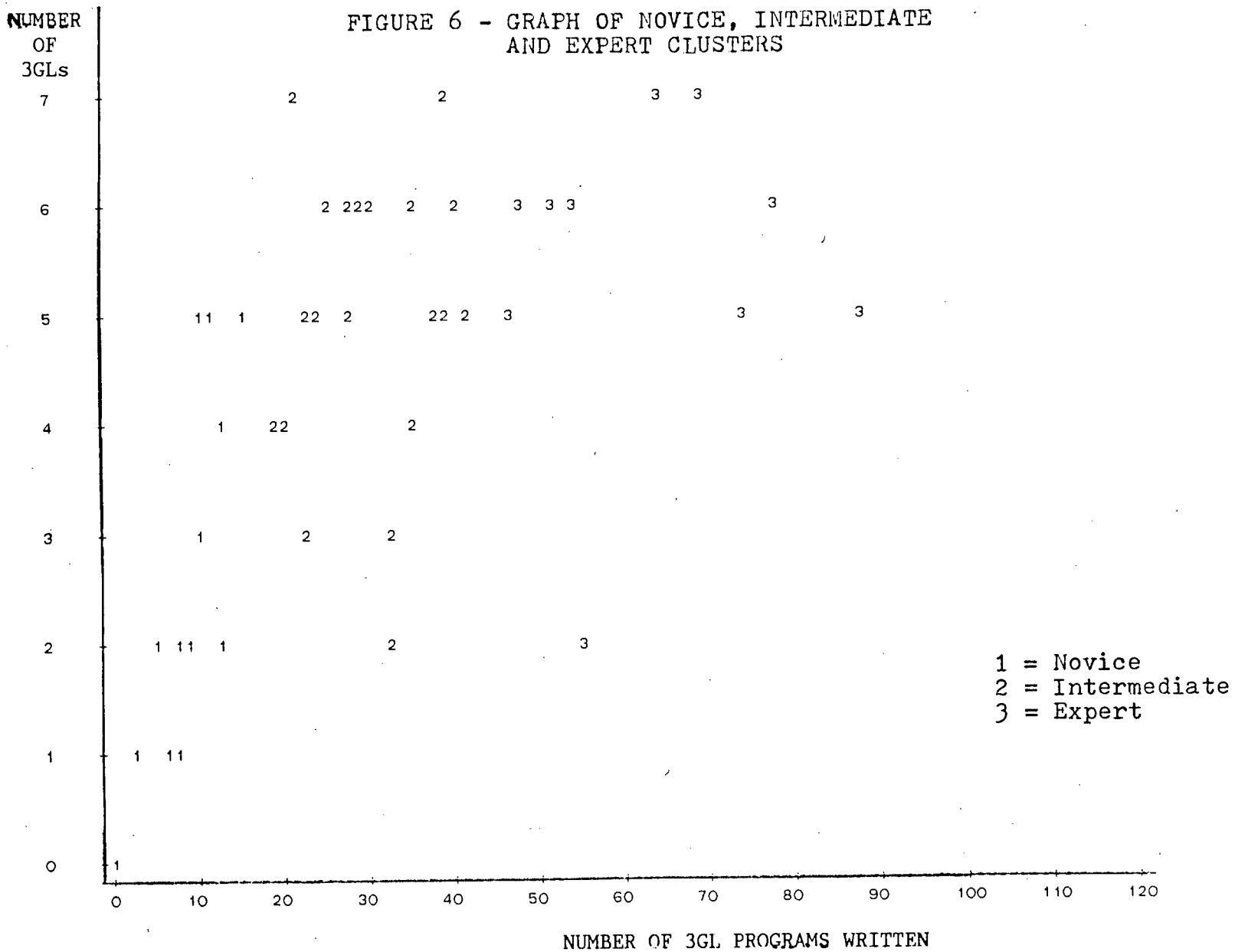
[50] Brian Everitt, Cluster Analysis (London: Heinemann Educational Books,1974), p.1.

analysis the loss of information which results from the grouping of individuals into clusters can be measured by the total sum of squared deviations of every point from the mean of the cluster to which it belongs. At each step in the analysis, union of every possible pair of clusters is considered and the two clusters whose fusion results in the minimum increase in the error sum of square are combined."[Everitt, p.15]

Outliers in the data have a negative effect on the cluster analysis because they form their own cluster and impede decomposition of other clusters. For this reason, subject number fifty-six, who had much more experience than any other subject, was deleted.

Figure 6 shows the clusters produced by Ward's method. The ones represent novices, twos intermediates, and threes experts. Comparison of the clusters produced by Ward's method with those obtained from the judges' scores, shows that forty-two of the fifty-seven subjects are classified the same way by both methods (this is true for both the classifications obtained from the judges' scores). This represents a seventy-four percent agreement. In addition, all of the experienced cluster, and nineteen of the twenty subjects in the novice cluster, produced by the cluster analysis are classified in the same way by the judges. The only real disagreement occurs between the novice-intermediate and intermediate-expert classifications, because cluster analysis produces a larger intermediate group. Thus, the cluster analysis seems to lend validity to the judges' ratings.

FIGURE 6 - GRAPH OF NOVICE, INTERMEDIATE AND EXPERT CLUSTERS



5.2.4 STATEMENT OF THE MODEL EQUATION

The hypotheses were tested empirically using the following model:

$$SCORE_{ijk} = \mu.. + EXPERIENCE_i + COMPLEXITY_j + (EXPERIENCE*COMPLEXITY)_{ij} + EDUCATIONAL\ INSTITUTION_k + \epsilon_{ijk} \quad \ldots\ldots\ldots\ldots (1.0)$$

where $\mu..$ is the overall mean

i is the level of the EXPERIENCE factor

j is the level of the COMPLEXITY factor

k is the level of the blocking factor

and ϵ_{ijk} is the random error.

5.2.5 OTHER VARIABLES USED IN THE ANALYSIS

Some of the subjects in the experiment had backgrounds which included experience with other types of software besides third generation languages. This experience could affect their performance on the FOCUS tests. Since a fourth generation language, such as FOCUS, is basically an integration of other types of software, such as a query language, report writer, screen painter, database management system, procedural language etc., individual experience with these other types of software might have an affect on how well a subject can learn FOCUS.

In order to account for possible confounding effects, the subjects' experience with various types of software were recorded, and were related to their performance on the FOCUS tests via a regression study. Before proceeding with the regression study two questions

needed immediate answers: How could software experience be categorized, and which software products fit into which categories? The first attempt resulted in a categorization of software experience into the following groups: nonprocedural languages, report writers, query languages, and fourth generation languages. These categories were not satisfactory because most of them were badly defined, and because they overlapped. For example, a fourth generation language could also be categorized as a query language or a nonprocedural language. Even though terms like query language and report writer are commonly used, there are no accepted definitions for any of these categories.

Some weak definitions have been given in the literature for these terms. For example, the National Bureau of Standards in the U.S., has defined a query language as a "language used to specify how database objects (items, entities, and relationships) are retrieved, manipulated (inserted, deleted, and modified) and how new objects are created." [51] Sometimes the inserting, deleting, and modifying aspects are not considered to be part of a query language, but, rather, make up a data sublanguage. Reisner defines a query language in the stricter sense, "A query language is a special purpose language for constructing queries to retrieve information from a database of information stored in a computer. It is usually intended to be used by people who are not professional programmers. Query languages are usually higher level languages with a fairly limited number of functions." [52] The problem with this definition is that languages like

---

[51]National Bureau of Standards, An Architecture for Database Management Standards, NSB Special Publication 500-86 (Washington: National Bureau of Standards, 1982) p.37.

[52]Reisner, "Human Factors Studies of Database Query Languages: A

SQL and QBE can no longer be considered query languages.

In order to make the categorization simpler, three mutually exclusive categories were created. The first category was query languages. Query languages were defined as languages used to retrieve and update information in a database. In this sense, query languages do not allow the user to select the location (column position) or appearance (insert commas, dollar signs, etc.) of fields in the report, rather the user must accept the default report. Query languages also lack the logic of procedural languages.

The second category was report writers. Report writers were defined as languages which had extensive formatting functions which could be used to produce simple and complex reports from a database or sequential file. This includes the ability to produce financial and statistical reports. In this sense, report writers have some programming logic, but lack good interactive query facilities.

The third category was fourth generation languages. As defined earlier, fourth generation languages include a query language, a report writer, a screen painter, a nonprocedural language, a database management system, and some procedural code for complex logic. For the purpose of this research, if software fell somewhere in-between a query language or report writer, and a fourth generation language, it was classified as a fourth generation language. This should not weaken the results because regression of test scores, with the first two categories, will show how using a query language or report writer affects using a fourth generation language. Regression of test scores with the last category will indicate how using multiple parts of a

---

[52](cont d) Survey and Assessment" p.14.

fourth generation language (or all the parts) will affect using the reporting commands of a fourth generation language.

The effect of having prior experience with other software tools, on the subject's ability to learn a 4GL, depends heavily on what prior tasks the subject has performed with the software tools. As it was not feasible to ask each subject what tasks he performed with the software, we could not determine how well the subject knew the particular tool. Therefore, the results of the regression are somewhat weakened by the fact that we only use aggregate experience data.

Software used by subjects is classified below in Table IV.

TABLE IV - OTHER SOFTWARE USED BY THE SUBJECTS

Query Language	Report Writer	Fourth Generation Language
SQL	COBOL REPORT	DBASE II and III
QUERY	WRITER	RBASE 5000
EDBS	MARK IV	KMAN
	RPG III	ORACLE
	IFPS	IMAGE
	SAS, GPSS	

5.3 MEASURE OF THE DEPENDENT VARIABLE

The subjects' ability to learn fourth generation language reporting commands was measured by their performance on either the simple or complex reporting test. Both the simple and complex tests were made up of three questions. Appendix 2 contains the simple test, Appendix 3 contains the complex test, and Appendix 4 contains the marking scheme used to score the two tests.

As can be seen, question One on the simple and complex tests are identical. This first question was intended to familiarize the student with a FOCUS exercise. As a result, question One was a very easy question, and, therefore, was only included in the overall score for the simple test. The overall score for the complex test included only the scores on questions two and three.

Question two on the complex test was more difficult than question two on the simple test, but both questions required the subject to produce a similar report on registration information. The two question Three's required the subject to produce similar reports, one being more complex than the other. Question Two and Three were really measuring the same thing within each test, and that was, the ability of the subjects to learn the fourth generation language commands (either simple or complex). But, as can be seen from the appendices, question three was much more comprehensive, and used many more FOCUS commands than question two.

Because question Three is very comprehensive, it could be used as a measure of the dependent variable, ability to learn a 4GL. The overall test score could also be used as a measure of the dependent variable. Questions one and two were not comprehensive enough to be

used as valid measures of 4GL learning ability. Regression analysis indicated that the scores achieved on question three, and overall score were highly correlated (R^2 = .7335). Since the overall score incorporates the scores achieved on question three, only the overall score will be used as a measure of the dependent variable.

The tests were marked by an independent FOCUS expert (an MIS professional) who was not involved in the research. An independent judge was used because someone involved in the research would have been less objective in his scoring.

5.4 PROCEDURE

5.4.1 PILOT TEST

Prior to the actual experiment, a pilot test was conducted to identify any weaknesses in the experimental materials, and to obtain practice in administering the experimental sessions. Eleven MBA students at UBC took part in the pre-testing, as part of a systems analysis course they were taking. One other MBA student was asked to take part because he had little knowledge of computers. The systems analysis students had varied amounts of prior programming experience. The students were randomly assigned to either the complex or simple reporting tasks.

During the experiment, subjects were isolated in a quiet room. For the majority of the time, the only other person in the room was the lab assistant. An IBM AT was used to run the fourth generation language. The pre-test sessions ran smoothly; all students finished the experiment in less than the three hours scheduled. The cumulative

time taken to complete the session varied from eighty-nine to one hundred and seventy-seven minutes.

Subjects were given forty-five minutes to complete the test. The subjects who wrote the simple test generally finished in less than forty-five minutes (only one of the six subjects took the maximum forty-five minutes). Scores on the simple test varied from fifty-seven to ninety-four percent, with four of the six subjects scoring above eighty percent. The subjects who wrote the complex test generally took all forty-five minutes (four of the five subjects) to complete the test. Scores on the complex test varied from sixty-one to eighty-seven percent, with four out of the five scoring seventy percent or higher. In order to achieve a wider range of scores, for data analysis purposes, it was decided to add two small sections to question one, and question three was made slighly longer (on both simple and complex tests). Forty-five minutes remained as the alloted time for both tests. The thesis results, which will be presented later, indicated that this forty-five minute time limit, along with the increased difficulty resulted in a much more difficult complex test.

While doing the sample problems some subjects complained that they were unable to understand the data structure used in the problems. For example, some did not know if the RETURNS field in the SALES file contained information about just one product, or was aggregate data by store. In order to clarify these problems, examples of data records were added to the file.

Subjects also experienced problems trying to decide what the proper order of the DEFINE and JOIN commands were, when the two commands were used in the same program. In order to help the subjects

understand the logic of these commands, problem number eight was later redesigned to involve both the DEFINE and JOIN commands.

Other than these two minor problems, the subjects had no other difficulties.

As a result of the subjects' comments, as well as comments made by other reviewers, several changes were made to the Report Generation Manual, after the pre-test sessions.

The following sections were deleted, because they were not directly relevant to the test, and, were time-consuming: Direct Operations, Includes and Excludes Tests, Testing Accumulated Values, Reports with no Verbs, Calculations.

In addition, the explanations for the PRINT, SUM and COUNT verbs were expanded because the students were occasionally confused as to which verb to use in a given situation. Examples were also added to some sections of the manual in order to clarify the purpose of certain commands.

Finally, the section describing the JOIN command was expanded because of subject confusion. This confusion was both verbalized, and obvious from the results of the pilot test.

5.4.2 THE ACTUAL EXPERIMENT

Each subject completed a three part session which lasted approximately three hours. The first part of the session involved learning a fourth generation language, the second involved practicing fourth generation language commands in a sample problem session, and the third tested the subject's knowledge of the fourth generation language which they had just learned. As mentioned earlier,the fourth

generation language used was FOCUS, the most popular fourth generation language on the market today. FOCUS was used because it has all of the characteristics of a fourth generation language which were enumerated in an earlier chapter of this thesis. FOCUS, an application generator, was used rather than a code generator, because application generators are more widely used in industry, and are more oriented towards end-users. The specific range of FOCUS reporting commands used in the tests were enumerated in Table II of this thesis.

The lab assistant began the sessions by giving a brief explanation of the purpose of the experiment (the lab procedures followed during the experiment are shown in Appendix 5). He explained that the main purpose of the experiment was to compare the ability of novices and experienced third generation language programmers to learn a fourth generation language. Next, the lab assistant discussed how important fourth generation languages are becoming in business. This was intended to reinforce the subjects' belief that they would learn something useful. Following this, the lab assistant discussed the importance of collecting the data for this thesis, and then, briefly, covered the sequence of events for the session. At this point, the lab assistant stressed that the session would take three hours, and possibly more. This was mentioned, in order to avoid losing disinterested students after the reading or practice stages of the experiment. It was hoped that once a student began, he would commit himself to finishing the session.

As mentioned before, the lab sessions were comprised of three main parts.

Part 1

The approximate duration of Part 1 was one-half to one and one-half hours. Subjects learned FOCUS commands used to generate reports, by reading an instruction manual adapted, by the author, from the PC/FOCUS Users Manual. This manual appears in Appendix 6. The subjects were encouraged to take as much time as necessary to read the manual,and to read it carefully, as it would save them time when they proceeded to the practice and test portions of the experiment. The lab experiments, in which the subjects were tested, were slightly different at the two institutions (UBC and BCIT). At BCIT, large numbers of people were run through the session at the same time. At UBC, subjects were tested individually. As a result, less control could be exercised over the subjects at BCIT, even though they were asked not to talk to one another during the session. This was one of the reasons for using educational institution as a blocking variable in the statistical analysis; the second reason being different educational background.

Part 2

The approximate duration of Part 2 was one-half to one and one-half hours. After reading the FOCUS manual, the subjects were given a set of eight practice problems (see Appendix 7) in order to practice the commands explained in the manual. The practice problems were based on a fictitious milk company's database, which were taken from the PC/FOCUS Operations Manual. A brief description of the database, the

description of the fields making up the database, and an example of database records were included with the practice problems. Subjects were asked to budget as close to an hour as possible for the practice problems. They were also provided with the solutions to the problems. Subjects were instructed to consult the solutions only after attempting the problem at least once, but were encouraged to refer to the solution if they were spending an excessive amount of time on any one problem.

Each of the sample problems forced the subject to use one or more of the most important FOCUS commands. Solutions to the problems were typed into the computer by the subject, who then executed them using FOCUS. This was continued iteratively until the subject answered correctly. The time taken to complete the problem session was recorded.

Part 3

The approximate duration of Part 3 was one hour. After completing the problem session, subjects were given a written test. The subjects were randomly assigned to either the simple or complex test. The tests involved creating course and subject reports from a university registration database, which had been created by the author. A forty-five minute time limit was imposed for the simple and complex tests. Subjects were advised when two or three minutes remained in the test.

Following the test, the subjects were required to fill out a questionnaire asking for personal data such as: number of years work

experience, educational level, knowledge of other 4GL's, prior use of report writers, query languages, data base management programs, computer languages known, number of programs written, number of years programming etc. The questionnaire is shown in Appendix 8. This data was used by the judges to rate the subjects as either novices or experts, and to test hypothesis number three, concerning the effect of other variables on the subject's ability to learn a fourth generation language.

5.5 ESTIMATION OF THE SAMPLE SIZE NEEDED

Before running the experiment itself, statistical analysis was performed to estimate the size of the sample needed. Two methods of estimating the sample size were used. The calculations are shown in Appendix 9. The results indicated that a sample size of 48 subjects would enable us to test the hypotheses with a sufficient level of confidence.

5.6 STATISTICAL METHODS USED

5.6.1 HYPOTHESIS ONE

Analysis of variance was used to determine if 3GL experience significantly affects the subjects' performance on tests of fourth generation languages. Specifically, one-sided t-tests were performed to determine if experienced third generation language programmers achieved higher mean scores than novices on simple and complex tests of fourth generation languages. The tested hyptheses were:

Ho: Experienced programmers' scores on simple 4GL tests will be equal to novices' scores on simple 4GL tests. (i.e. $\mu_{21} = \mu_{11}$)
Ha: Experienced programmers' scores on simple 4GL tests will be greater than novices' scores on simple 4GL tests. (i.e. $\mu_{21} > \mu_{11}$)

and

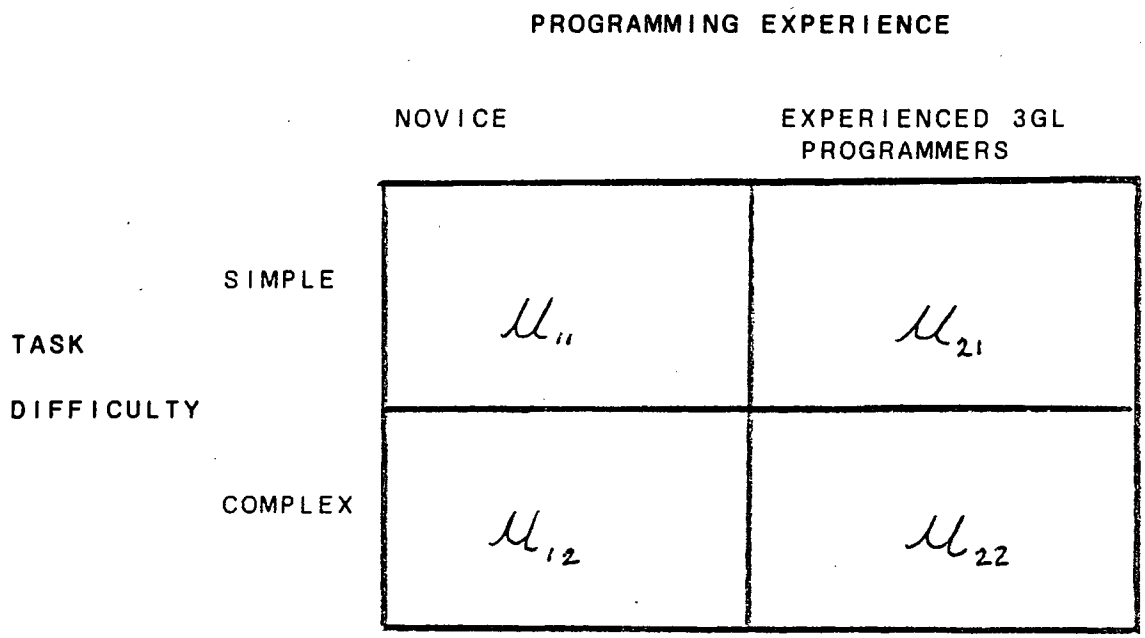
Ho: Experienced programmers' scores on complex 4GL tests will be equal to novices' scores on complex 4GL tests. (i.e. $\mu_{22} = \mu_{12}$)
Ha: Experienced programmers' scores on complex tests will be greater than novices' scores on complex 4GL tests. (i.e. $\mu_{22} > \mu_{12}$)

To test these hypotheses we used one-sided t-tests (at the 5% and 10% levels of significance). With reference to Figure 7, if we let L equal the difference between experienced and novice scores (e.g. $\mu_{21} - \mu_{11}$), then the critical value t* is calculated as follows: $t^* = (L-0)/S(L)$ where $L = Y_{21} - Y_{11}$, and $S^2(L) = MSE\ (1/n_{ij} + 1/n_{i'j'})$. The t table value will have n_T - ab degrees of freedom where n is the sample size, a and b are the number of levels of the first and second factor.

Once t* is calculated we would compare it with the t-value obtained from a t-distribution table for a given level of significance. If $t^* \leq t$ we will accept Ho, otherwise if $t^* > t$ we will reject Ho and accept Ha.

Regression analysis was also used to support the analysis of variance results, and to determine the numerical relationship between the subjects' experience with 3GLs, and their ability to learn a 4GL.

FIGURE 7 - EXPERIMENTAL DESIGN WITHOUT BLOCKING



For this analysis, the dependent FOCUS test score (OV) was regressed with the independent variable, judges' mean experience rating (SCA) (measuring the subjects' experience with 3GLs). The model is as follows:

$$\text{Test Score}_i = \beta_0 + \beta_1 \text{Experience Rating}_i + \epsilon_i \quad \ldots\ldots\ldots (1.1)$$

We examined both the significance of the relationship (R^2), the direction of β_1, and the p-value for the Experience Rating variable. Hypothesis number one predicts $\beta_1 > 0$. The above analysis was repeated for both the simple and complex tests. The two tests could not be combined in the same analysis because a certain score on the complex test indicated a completely different performance than the same score on the simple test. Scores on the simple test were significantly higher.

5.6.2 HYPOTHESIS TWO

Analysis of variance was used to determine if the novices' mean drop in scores between the simple and the complex test was greater than the experienced 3GL programmers mean drop in scores. If the (EXPERIENCE * COMPLEXITY) interaction term in our model is significant, we can conclude that the difference in test score, between the simple and complex test, has changed significantly either for the novices or the experienced 3GL programmers. But, specifically, we would like to test if novice scores have decreased by a larger number. We can test this using a one-sided t-test.

Ho: $(\mu_{11} - \mu_{12}) = (\mu_{21} - \mu_{22})$

Ha: $(\mu_{11} - \mu_{12}) > (\mu_{21} - \mu_{22})$

where $t^* = (L-0)/ S(L)$ where $L = (Y_{11} - Y_{12}) - (Y_{21} - Y_{22})$ and $s(L) = MSE \Sigma\Sigma c_{ij^2}/n_{ij}$. If Ho is accepted, we would conclude that there is no evidence indicating that novice programmers' scores decrease more than experienced 3GL programmer scores.

5.6.3 HYPOTHESIS THREE

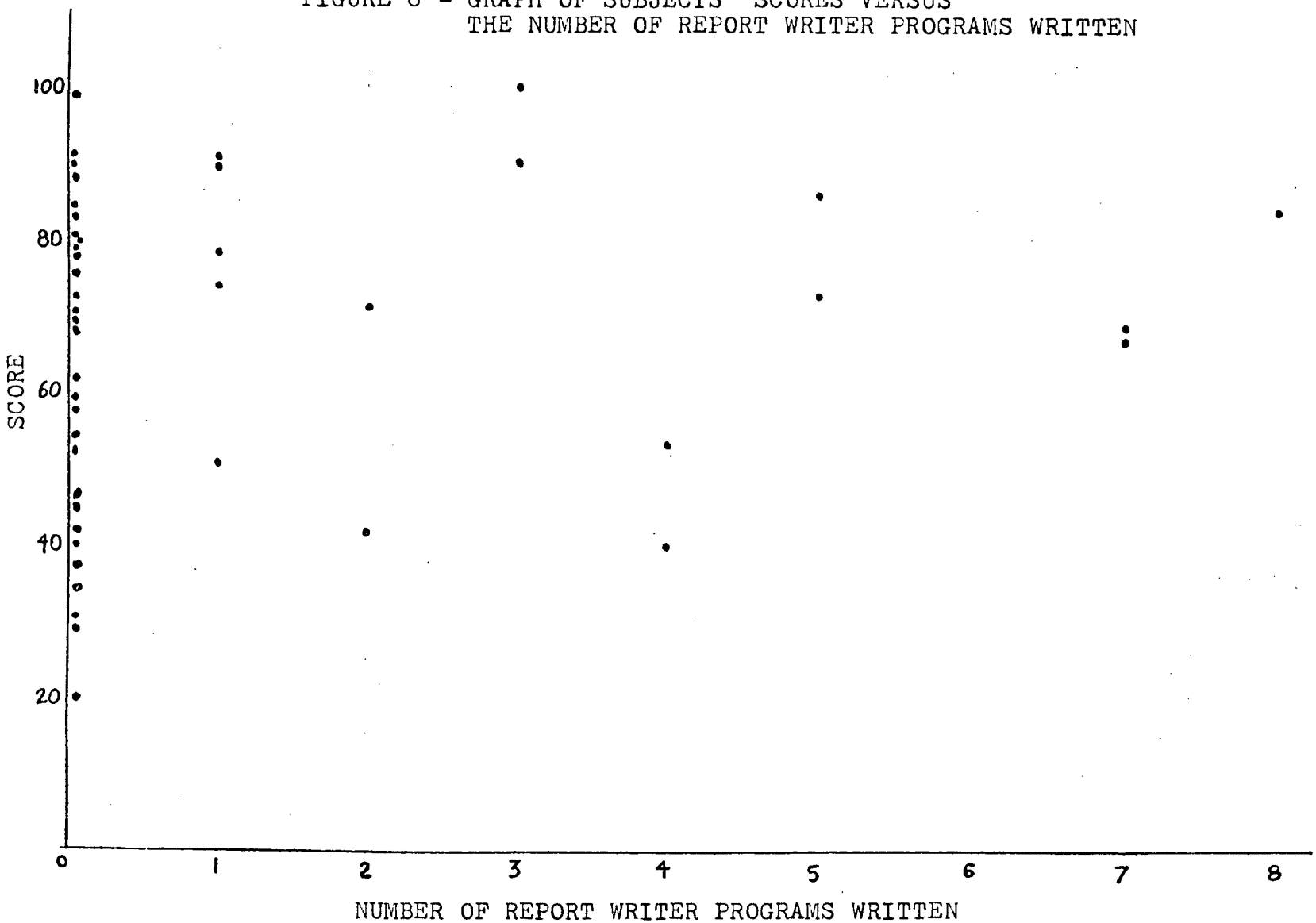
Simple and multiple regression analyses were used to test hypothesis three: that other variables are positively related to the subjects' test scores. Scatter plots of the number of query language, report writer, and 4GL programs written versus test scores were examined before performing the regressions to ensure that enough data was available to proceed with the regression. Figure 8 shows the graph of subjects' scores versus the number of Report Writer programs written. The scatter plots of subjects' scores versus the number of query language programs written, and the number of fourth generation language programs written are very similar. It is obvious that there is a shortage of data for all the variables, but the data for the query language programs written is most scarce. Only four subjects had used query languages before and no one had written more than two query language programs. The query language variable was not examined because of the shortage of data. The following models were analyzed:

$$\text{Test Score} = \beta_0 + \beta_1 \text{Report Writer} + \epsilon \quad \ldots\ldots\ldots\ldots\ldots\ldots (1.2)$$

$$\text{Test Score} = \beta_0 + \beta_1 4GL + \epsilon \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1.3)$$

$$\text{Test Score} = \beta_0 + \beta_1 \text{Report Writer} + \beta_2 4GL + \epsilon \quad \ldots\ldots\ldots\ldots (1.4)$$

FIGURE 8 - GRAPH OF SUBJECTS' SCORES VERSUS
THE NUMBER OF REPORT WRITER PROGRAMS WRITTEN



Report Writer, and 4GL, are dummy variables, with the following meaning: if the variable=1 then the subject has used this tool before, if the variable=0 then the subject has no experience with the tool. The above regressions were repeated for both simple and complex tests.

From the results, we looked for overall significance of the relationship (R^2, F-statistic), and a positive β_1 value. A positive β_1 value indicated that, as hypothesized, a greater amount of experience with the tool results in higher 4GL test scores.

5.7 VARIABLES USED IN THE ANALYSIS

In the course of the research, the variables shown in Table V were used in the analysis of the hypotheses. The data collected from the subjects appears in Appendix 10.

TABLE V - VARIABLES USED IN THE ANALYSIS

VARIABLE NAME	TYPE OF VARIABLE
1. Educational Institution(EI)	Dummy variable (UBC=1, BCIT=0)
2. Previous Related Full-Time Work Experience (PWE)	Dummy variable (YES=1, NO=0)
3. Years of Programming Experience at Work(YPEW)	Continuous variable (=percent of time * no of years)
4. MBA student (MBA)	Dummy variable (YES=1, NO=0)

5. Computer Systems student (SS)	Dummy varialble (YES=1, NO=0)
6. Other student (OS)	Dummy variable (YES=1, NO=0)
7. Simple/Complex Task (COMPLEXITY)	Dummy variable (Simple=1, Complex=0)
8. Previous 3GL Experience (E3GL)	Dummy variable (YES=1, NO=0)
9. Number of 3GLs Known (N3GL)	Continuous variable
10. Number of 3GL Programs Written (N3GLW)	Continuous variable
11. Number of Report Writer Programs Written (RWPW)	Continuous variable
12. Number of Query Language Programs Written (QLPW)	Continuous variable
13. Number of 4GL Programs Written (N4GLW)	Continuous variable
14. Total time of the Subject's Session (TTIME)	Continuous variable
15. Reading time (RT)	Continuous variable
16. Problem time (PT)	Continuous variable
17. Test time (TT)	Continuous variable
18. Overall Score on FOCUS Test (OV)	Continuous variable
19. Score on Q1 (Q1)	Continuous variable

20. Score on Q2 (Q2)	Continuous variable
21. Score on Q3 (Q3)	Continuous variable
22. Novice/Experienced 3GL Classification (EXP)	Dummy variable (Novice=0 Exp3GL=1)
23. Mean experience score of the three judges (SCA)	Continuous variable (Scale of 1 to 7)

6. RESULTS

6.1 HYPOTHESIS ONE

The analysis of variance was performed for model 1.0. The ANOVA table is shown below in Table VI.

TABLE VI - ANOVA TABLE FOR MODEL 1.0

Source of variation	SS	df	MS	F-value	PR > F
COMPLEXITY	4341.4	1	4341.4	12.54	.0011
EXPERIENCE	29.8	1	29.8	0.09	.7709
(COMP*EXP)	673.8	1	673.8	1.95	.1712
EDUCATIONAL INSTITUTION	319.3	1	319.8	0.92	.3430
ERROR	13159.6	38	346.3		

The ANOVA table indicates that Educational Institution has no effect on test scores. Subjects from both Educational Institutions did equally well. The table does indicate that complexity is an important factor, as was intended. Scores on the complex test are significantly different from scores on the simple test (p = .0011). The COMP*EXP interaction term indicates some weak interaction between complexity and experience. This will be discussed later under hypothesis two.

The ANOVA table indicates that Experience is not an important factor when the mean scores of simple and complex tests are combined. But, we also need to determine if 3GL experience has an effect on test scores for the simple test only. This must also be repeated for the complex test. We can test the hypothesis that experienced 3GL programmers achieve higher mean scores than novices, using the t-test described earlier, in the Method section.

First for the simple test, using the one-sided t-test described earlier, t*=1.45. The critical t-value at .05 level of significance is t[.95,39]= 1.684. Since t* is not larger than t we cannot reject Ho at the .05 level of significance. The critical t-value at the .10 level of significance is t[.90,39] = 1.303. Since t* = 1.45 is greater than 1.303 we can reject Ho, and conclude that 3GL experience is an important factor for simple tests at the .10 level.

For the complex test, t*= -.52. This is not larger than the critical t-value 1.684. Therefore we cannot reject Ho at the .05 level. The same conclusion is reached at the .10 level. This indicates that 3GL experience is not an important factor for the complex test.

The scatter plots of test scores versus the judges' mean experience ratings, for simple and complex tests, are shown in Figures 9 and 10. The scatter plot for the simple test shows an upward trend, whereas the scatter plot for the complex test is random. The results of regresson 1.1 score versus 3GL experience for the simple test, are as follows: R^2=.1318, p-value of .028, and β_1 = 4.4. The results for the complex test are R^2=.0015, p-value=.42, and β_1=0.46. These regression results support the results obtained from the analysis of

FIGURE 9 - GRAPH OF SUBJECT SCORES VERSUS MEAN OF THE JUDGES' RATINGS FOR THE SIMPLE TEST

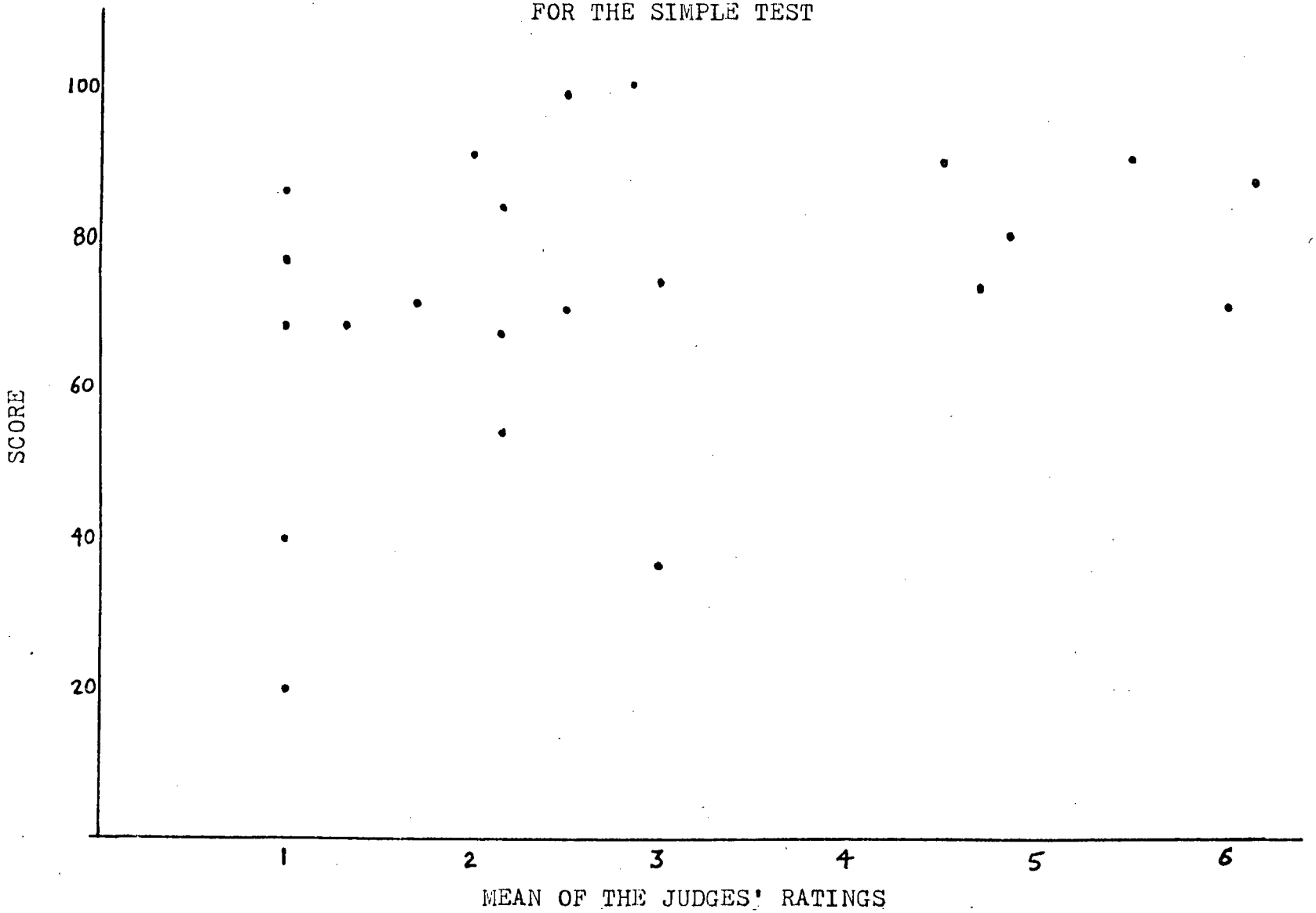
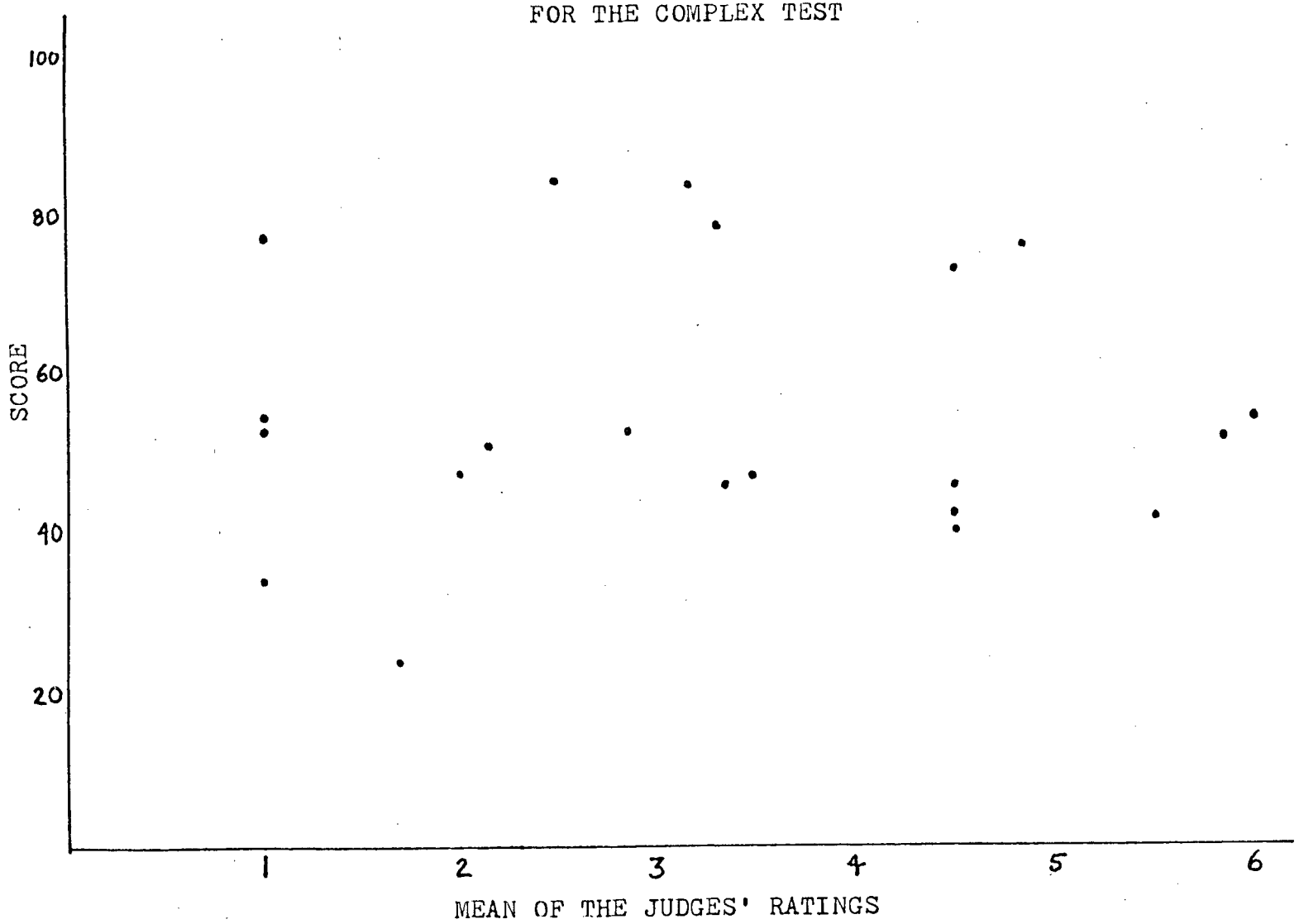


FIGURE 10 - GRAPH OF SUBJECTS' SCORES VERSUS MEAN OF THE JUDGES' RATINGS FOR THE COMPLEX TEST



variance for hypothesis one. The regression for the simple test is significant, again indicating that experience with 3GLs affects simple 4GL test performance. The positive β_1 value indicates that greater experience is correlated with higher simple test scores, as hypothesized. The results of the complex test regression again indicate that the experience-score relationship is not significant for the complex test.

6.2 HYPOTHESIS TWO

We can directly test hypothesis number two to see if novices' scores drop by more than experienced 3GL programmer scores. We will use a .10 level of significance. In order to accept Ha, that novice scores decrease more than experienced 3GL programmers' scores , t* will have to be larger than $t[.90, n_T - ab]$ which is 1.303. We determined that t* = -1.43, which is not larger than 1.303. Therefore we cannot reject Ho.

Therefore, the evidence indicates that we cannot conclude that novices' scores drop by more than experienced programmers' scores.

6.3 HYPOTHESIS THREE

Table VII presents the results of the regression analysis. We shall use the term Report Writ as a short form for the dummy variable Report Writer. The table indicates that 4GL is the only significant variable. The table shows the results of both the simple regressions and the multiple regressions. The multiple regression results are presented at the bottom of the table.

TABLE VII - TABLE OF REGRESSION STATISTICS FOR HYPOTHESIS 3

Simple/ Complex task	Dependent Variable	Independent Variable	P-value	R	Value of B
Simple	Score	Report Writ	.2065	.0259	6.16
Simple	Score	4GL	.0632	.0874	11.82
Complex	Score	Report Writ	.3296	.0073	-3.21
Complex	Score	4GL	.3468	.0058	2.56
Simple	Score	Report Writ	.1920		0.19
		4GL	.4484		0.92
Complex	Score	Report Writ	.3665		2.55
		4GL	.0068		22.4

More data would be needed to be certain of the results on report writer, and 4GL experience. The raw data collected shows that only nineteen out of the fifty-seven subjects had used report writers (no one had written more than sixteen programs), and only twenty-four out of fifty-seven had previously used a fourth generation language. These are both small samples.

Results indicate that 4GL is a significant variable. The fact that students, with previous 4GL experience do significantly better on the FOCUS 4GL tests is not surprising. The evidence we have indicates that we have to reject the hypothesis that experience with report writers leads to higher 4GL test scores.

6.4 SUMMARY OF THE PERFORMANCE OF THE SUBJECTS

Figure 11 and Table VIII summarize the performance of each of the four treatments, as well as the number of subjects in each treatment. Table VIII and Figure 11 both indicate that novice scores are much more variable than experienced programmer scores. The data strongly suggests that some novices performed poorly on both tests, while experienced programmers were more consistent.

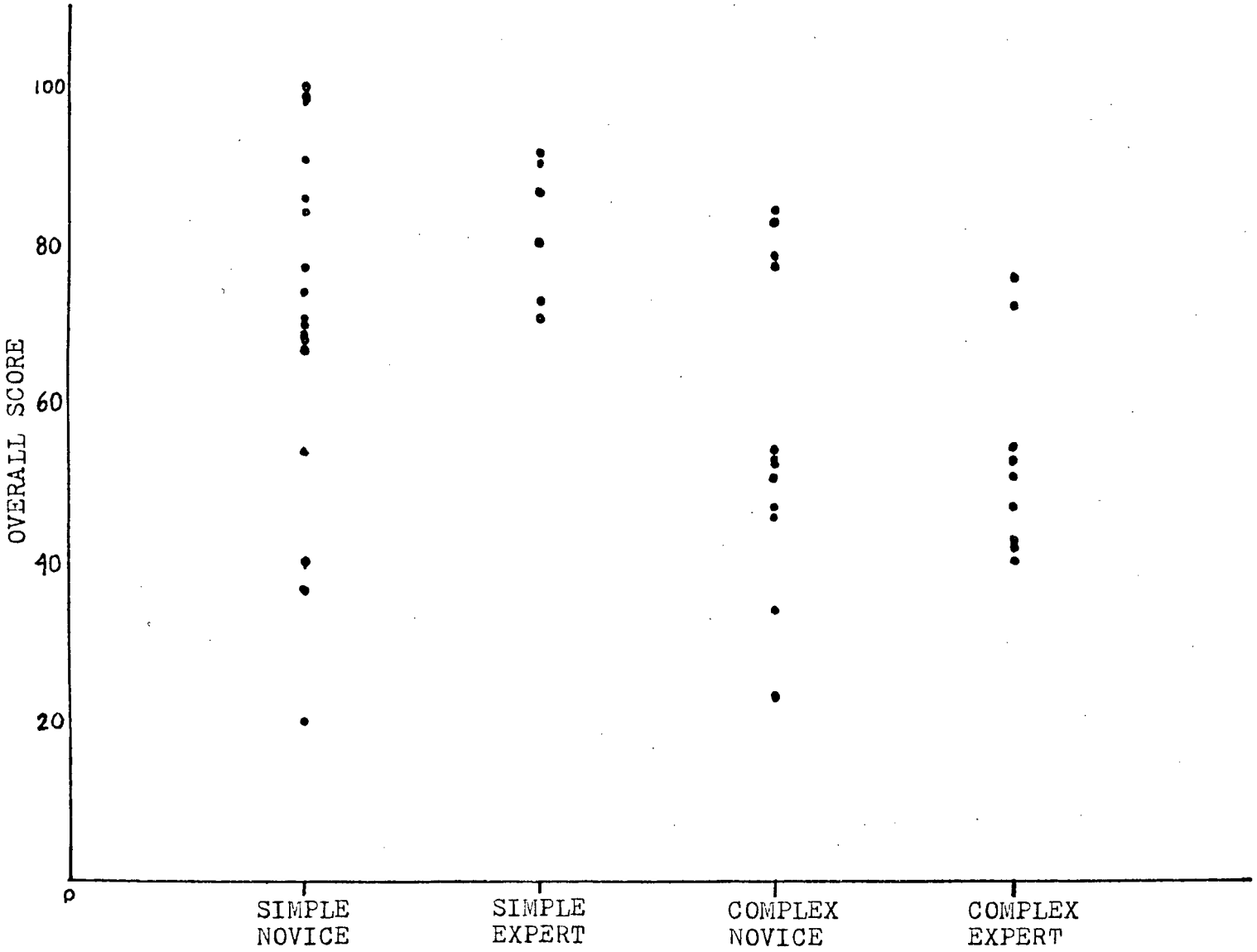
All the scores recorded on the complex test were low indicating that the test was very difficult, or, that too little time was provided for the subjects to finish the test. This could explain why there was little difference in the scores achieved by novices and experienced programmers on the complex test.

We also note in Figure 11 a break in scores between the lower scorers and the higher scorers. This break indicates that some subjects could learn 4GLs easily, while others had a lot of

TABLE VIII - MEANS, STANDARD ERRORS, AND NUMBER OF OBSERVATIONS FOR EACH TREATMENT

TASK DIFFICULTY	PROGRAMMING EXPERIENCE: NOVICE	PROGRAMMING EXPERIENCE: EXPERIENCED 3GL
SIMPLE	$\bar{X}$ = 69.1 σ = 22.3 N = 16.0	$\bar{X}$ = 82.0 ϵ = 8.7 N = 6.0
COMPLEX	$\bar{X}$ = 56.9 σ = 19.6 N = 12.0	$\bar{X}$ = 52.6 ϵ = 13.3 N = 9.0

FIGURE 11 - GRAPH OF SUBJECT SCORES VERSUS EACH OF THE FOUR TREATMENTS



difficulty. This facility with 4GLs is not solely dependent on previous 3GL experience as can be seen from the graph. There seems to be another factor missing here, which would explain why some subjects learn 4GLs more easily than others.

7. DISCUSSION OF THE RESULTS

Though not all the hypotheses were accepted when tested, some very interesting results were obtained from the analysis.

Before discussing the results, some weaknesses of the methodology, which could have affected the results, should be discussed. Because the subjects were only given approximately an hour to read the FOCUS commands, and one hour to run through the problem session, some question remains as to the validity of the measure of the dependent variable, ability to learn a fourth generation language. On the one hand, in a work setting, employees are given a number of days to familiarize themselves with the FOCUS commands. On the other hand, the two to three hour learning session given the students, allowed them to learn almost all of the FOCUS reporting commands. Because the syntax is so English-like, it takes little time to learn FOCUS.

Because the tasks assigned to students were relatively short, some critics might question the external validity of the results. The problems assigned to the subjects were typical of problems faced by business professionals. But these problems are more simplistic than those handled by information system professionals. A more realistic problem for an information system professional would involve building, maintaining, and reporting from a database. Getting the subjects to commit more time to complete a more complex task would have been a problem. The three hour session was already long.

In this study, only reporting tasks were examined, yet the novices showed that they handled more complex tasks no more poorly than the experienced programmers.

The final weakness in our methology is that no real "expert" programmers were used in the experiment. Rather, experienced student programmers were used. If professional programmers had been used they might have done better than the novices. Still, a few of the subjects had worked as programmers, and the rest of the experienced subjects were only one year away from being in the job market where they would be considered professional programmers.

Notwithstanding the above problems, the results are intriguing. The problems are a realistic representation of what some marketing, or finance professionals will be faced with in the near future.

The results of hypotheses two and three will be discussed first as they are not as important as the results of hypothesis one.

The results of hypothesis number two were not what we expected. We expected novices' scores to decrease by more than experienced programmers' scores, when progressing from the simple to the complex test. The results can be partially explained by the higher score recorded by experienced programmers on the simple test. When we were setting up the tests, we expected experienced programmers to perform much better than novices on the complex test. As a result, the second hypothesis would have measured a much bigger drop for novices. But, because experienced programmers scored higher than novices on the simple test, but no better on the complex test, the results of hypothesis number two became less meaningful than originally expected.

Because of the lack of data, few meaningful results were obtained for hypothesis number three. Only the fact that knowing one 4GL helps, when learning another, can be safely concluded.

Concerning hypothesis number one, the results of the simple test indicate that experienced programmers outperform novices. The simple commands such as PRINT, BY, and ON involving control breaks, are better handled by the experienced programmers. These commands are familiar to experienced third generation language programmers, therefore, they can transfer previous skills to a fourth generation application involving simple reporting commands. As a result, their scores are higher than novices' who have never been exposed to these commands.

Results of the complex test do not support the hypothesis that experienced programmers can outperform novices on complex 4GL tests. Some people may argue that these results are due to an excessively hard test combined with too short a testing time (45 minutes). Their argument would be that the test was too hard for all the subjects, and therefore the results are meaningless. Thirty percent of the subjects who wrote the complex test scored below fifty percent. Also, 16 out of the 29 subjects who wrote the complex test did not finish writing the test before the 45 minute time limit had expired. This data could indicate that many of the subjects found the test too long or too hard.

Based on the fact that the experienced programmers' scores on the complex test were less variable than the novices' scores, we might conclude that if a bigger sample was taken the experienced programmers would have outperformed the novices. I believe the results may be partly due to the difference between the simple and complex 4GL commands. The more complex 4GL reporting commands such as JOIN, SUBHEAD, and DEFINE used in the complex test are semantically very

different from anything encountered in a third generation language. Therefore, the experienced third generation language programmer cannot transfer any previous skills into this complex 4GL environment. These very complex commands were the ones which caused the most problems for experienced 3GL programmers. As a result, the experienced 3GL programmer has little or no advantage over a novice in a complex 4GL reporting application. The fact that experienced programmers' scores were less variable than novices' scores can be explained by the fact that a few simpler commands were used in the complex test. As we have already concluded, experienced programmers have an advantage over novices in simple 4GL reporting applications. Possibly, experienced programmers score no better than novices because they have no more experience with the different 4GL semantics than novices do.

The results of this research indicate that experienced third generation language programmers would be preferred over novices when an application involves simple 4GL commands. For complex applications, novices performed as well as experienced third generation language programmers, but the results were not conclusive. Several possible explanations were advanced to try to explain the results. More testing is needed to determine if any of the explanations is correct. Further testing should also be conducted by other researchers with other fourth generation languages, and with other tasks to see if the same results are obtained.

BIBLIOGRAPHY

Abbott, Jack L. "A Comparison of Five Database Management Programs" Byte, 8, No.5(1983), pp.220-228.

Brooks, Ruven. "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology" Communications of the ACM, 23(1980), pp.207-213.

Brooks, Ruven. "Using a Behavioral Theory of Program Comprehension in Software Engineering" IEEE Third International Conference on Software Engineering 1978. Long Beach,CA.: IEEE,1978, pp.196-201.

Cardenas, Alfonso F., and William P. Grafton. "Generators: Challenges and Requirements for New Applications" Proceedings of the National Computer Conference 1982. Montvale,N.J.: AFIPS,1982, pp.343-349.

Chamberlain, D.D., Astrahan, M.M., et al. "SEQUEL 2: A Unified Approach to Data Definition, Manipulation and Control" IBM Journal of Research and Development, 20(1976), pp.560-574.

Chapin, Ned. "Software Maintenance with Fourth Generation Languages" ACM Sigsoft Software Engineering Notes, 9, No.1(1984), pp.41-42.

Chase, William G., and Herbert A. Simon. "Perceptions in Chess" Cognitive Psychology, 4(1973), pp.55-81.

Chrysler, E. "Some Basic Determinants of Computer Programmming Productivity" Communications of the ACM, 21(1978), pp.472-483.

Chrysler, E. "The Impact of Program and Programmer Characteristics on Program Size" Proceedings of the National Computer Conference 1978. Montvale,N.J.: AFIPS,1978, pp.581-587.

Cobb, Richard H. "In Praise of 4GLs" Datamation, 31, No.14(1985), pp.90-96.

Coble, D.F. "Fourth Generation Languages Will Impact Productivity - If..."Data Management, 20, No.7(1982), pp.29-32.

Codd, E.F. "Relational Databases: A Practical Foundation for Productivity" Communications of the ACM, 25(1982), pp.109-117.

Cu-Uy-Gam, Miriam. "Do-it-yourself is on the way for system development" Computing Canada;Software Report,(May 1985),p.9.

Data Decisions. "System Software Survey: User's Favourite Disks" Datamation, 30, No.20(1984), pp.85-138.

Digital Consulting Associates. The 1984 National Data Base and Fourth Generation Language Symposium. Wakefield,MA.: Digital Consulting Associates,1984.

DuBoulay, B., and T. O'Shea. "Teaching Novices Programming" Human Interations With Computers, ed H.T. Smith and T.R.G. Green. New York: Academic Press,1980, pp.147-200.

Duncker, K. "On Problem Solving" Psychological Monographs, 58, No.5(1945), pp.1-112.

Dzida, W., Herda, S. et al. "User Perceived Quality of Interactive Systems" IEEE Transactions on Software Engineering, 4(1978), pp.270-276.

EDP Analyzer. Special Report: Fourth Generation Languages and Prototyping. Vista,CA.: Canning publications,1984.

Eisenbach, S., and C. Sadler. "Declarative Languages: An Overview" Byte, 10, No.8(1985), pp.181-197.

Elder, Marvin. "SALVO - A Fourth Generation Language for Personal Computers" Proceedings of the National Computer Conference 1984. Montvale,N.J.: AFIPS,1984, pp.563-566.

Everitt, Brian. Cluster Analysis. London: Heinemann Educational Books,1974.

"Fourth Generation Languages enter the dp mainstream despite some resistance" Computing Canada: Software Report, (May 1985), p.6.

Garry, Ralph, and Howard L. Kingsley. The Nature and Conditions of Learning. Englewood Cliffs,N.J.: Prentice-Hall,1970.

Goodman, Aaron M. "Application Generators at IBM" Proceedings of the National Computer Conference 1982. Montvale,N.J.: AFIPS,1982, pp.361-362.

Gordon, J.D., Salvadori, A., and C.K. Capstick. "An Empirical Study of COBOL Programmers" INFOR, 15(1977), pp.229-241.

Gould, John D. "Some Psychological Evidence on How People Debug Computer Programs" International Journal of Man-Machine Studies, 7(1975), pp.151-182.

Green, T.R.G., Sime, M.E., and M.J. Fitter. "The Art of Notation" Human Interactions with Computers, ed H.T. Smith and T.R.G. Green. New York: Academic Press,1980, pp.221-251.

Grochow, Jerrold M. "Application Generators: An Introduction" Proceedings of the National Computer Conference 1982. Montvale,N.J.: AFIPS,1982, pp.391-392.

Harel, Elie C., and Ephrain R. McLean. "The Effects of Using a Nonprocedural Computer Language on Programmer Productivity" MIS Quarterly, 9, No.2(1985), pp.109-120.

Holtz, D.H. "A Nonprocedural Language for On-Line Applications"

Datamation, 25, No.4(1979), pp.167-176.

Horowitz, Elie, Kemper, Alfons, and Balaji Narasimhan. "A Survey of Application Generators" IEEE Software, 2, No.1(1985), pp.40-54.

Jenkins, Milton A. "Surveying the Software Generator Market" Datamation, 31, No.17(1985), pp.247-261.

Johnson, James R. "A Prototypical Success Story" Datamation, 29, No.11(1983), pp.251-256.

Johnson, Jan. "MAPPER Goes Micro" Datamation, 29, No.11(1983), pp.62-66.

Kelley, J.F. "An Iterative Design Methodology for User-Friendly Natural Language Office Information Applications" ACM Transactions on Office Information Systems, 2(1984), pp.26-40.

Kennedy, T.C.S. "Some Behavioural Factors Affecting the Training of Naive Users of an Interactive Computer System" International Journal of Man-Machine Studies, 7(1975), pp.817-834.

Kowalski, R. "Logic Programming" Byte, 10, No.8(1985), pp.161-177.

Kull, David. "Non Procedural Languages: Bringing up the Fourth Generation" Computer Decisions, 15, No.13(1983), pp.154-162.

Landaver, T.K., Galotti, K.M., and S. Hartwell. "Natural Command Names and Initial Learning: A Study of Text-Editing Terms" Communications of the ACM, 26(1983), pp.495-503.

Laughery Jr., K. Ronald, and Kenneth R. Laughery Sr. "Human Factors in Software Engineering: A Review of the Literature" The Journal of Systems and Software, 5(1985), pp.3-14.

Leavenworth, Burt M., and Jean E. Sammet. "An Overview of Nonprocedural Languages" IBM Research Report, RC4685. Gaithersburg,MA.: IBM,1974.

Lukac, Eugene G. "The Impact of a 4GL on Hardware Resources" Datamation, 30, No.16(1984), pp.105-114.

Mandell, Steven L. Computers and Data Processing: Concepts and Applications. New York: West Publishing Company,1985.

Martin, James. Application Development Without Programmers. Toronto: Prentice-Hall,1982.

Martin, James. Fourth Generation Languages. Lancaster: Savant Institute,1983.

Mayer, Richard E. "The Psychology of How Novices Learn Computer Programming" ACM Computing Surveys, 13(1981), pp.121-141.

McDonald, Nancy H., and John P. McNally. "Query Languages Feature Analysis by Usability" Computer Languages, 7(1982), pp.103-124.

McKeithen, Katherine B. et al. "Knowledge Organization and Skill Differences in Computer Programmers" Cognitive Psychology, 13(1981), pp.307-325.

Miara, Richard J., Musselman, Joyce A., et al. "Program Indentation and Comprehensibility" Communications of the ACM, 26(1983), pp.861-867.

Miller, Boulton B. "Fourth generation languages and personal computers" Proceedings of the National Computer Conference 1984. Montvale,N.J.: AFIPS,1984, pp.555-559.

Miller, Lance A., and Curtis A. Becker. "Programming in Natural Language" IBM Research Report, RC5137. Yorktown Heights: IBM,1974.

Miller, Lance A., and John C. Thomas Jr. "Behavioral Issues in the Use of Interactive Systems" International Journal of Man-Machine Studies, 9(1977), pp.509-536.

Moran, Thomas P. "An Applied Psychology of the User" ACM Computing Surveys, 13(1981), pp.1-11.

Munnecke, Thomas. "A Linguistic Comparison of MUMPS and COBOL" Proceedings of the National Computer Conference 1980. Montvale,N.J.: AFIPS,1980, pp.723-729.

National Bureau of Standards. An Architecture for Database Management Standards, NBS Special Publication 500-86. Washington: National Bureau of Standards,1982.

Neter, John, Wasserman, William, and Micheal H. Kutner. Applied Linear Statistical Models. Homewood, Ill.: Richard D. Irwin Inc.,1985.

Nicol-Griffith, Mike. MAPPER was the First User Command Language. Montreal: Canadian Pacific Consulting Services,1983.

Nicol-Griffith, Mike. User-Driven Computing at Canadian Pacific Consulting Services Ltd. - A Case Study. Montreal: Canadian Pacific Consulting Services,1985.

Osgood, C.E. Method and Theory in Experimental Psychology. New York: Oxford,1953.

Paxton, A.L., and E.J. Turner. "Human Factors and Novice Computer Users" International Journal of Man-Machine Studies, 20(1984), pp.137-156.

Petrick, S.R. "On Natural Language Based Computer Systems" IBM Journal of Research and Development, 20(1976), pp.314-325.

Prywes, N.S., Shastry, S. and A. Pnueli. "Use of a Nonprocedural Specification Language and Associated Program Generator in Software Development" ACM Transactions on Programming Languages and Systems, 1(1979), pp.196-217.

Read, Nigel S., and Douglas L. Harmon. "Assuring MIS Success" Datamation, 27, No.29(1981), pp.109-120.

Read, Nigel S., and Douglas L. Harmon. "Readers' Forum: Language Barrier to Productivity" Datamation, 29, No.2(1983), pp.209-212.

Reisner, Phyllis, Chamberlain, Donald D., and Raymond F. Boyce. "Human Factors Evaluation of Two Data Base Query Languages: Square and Sequel" Proceedings of the National Computer Conference 1975. Montvale,N.J.: AFIPS,1975, pp.447-452.

Reisner, Phyllis. "Human Factors Studies of Database Query Languages: A Survey and Assessment" ACM Computing Surveys, 13(1981), ppl3-31.

Reisner, Phyllis. "Use of Psychological Experimentation as an Aid to Development of a Query Language" IEEE Transactions on Software Engineering, 3(1977), pp.218-229.

Sammet, Jean E. Programming Languages: History and Fundamentals. Englewood Cliffs,N.J.: Prentice-Hall,1969.

SAS Institute Inc. SAS User's Guide: Statistics, Version 5 Edition.Cary, NC: SAS Institute Inc.,1985.

Schmidt, Joachim W. "Some High Level Constructs for Data of the Type Relation" ACM Transactions on Data Bases, 2(1977), pp.247-261.

Sheil, B.A. "The Psychological Study of Programming" ACM Computing Surveys, 13(1981), pp.101-120.

Shneiderman, Ben. Software Psychology: Human Factors in Computer and Information Systems. Toronto: Little, Brown and Co.,1980.

Shneiderman, Ben. "Measuring Computer Program Quality and Comprehension" International Journal of Man-Machine Studies, 9(1977), pp.465-478.

Shneiderman, Ben. "Exploratory Experiments in Programmer Behavior" International Journal of Computer and Information Sciences, 5, No.2(1976), pp.123-143.

Shneiderman, Ben. "Improving the Human Factors Aspect of Database Interactions" ACM Transactions on Database Systems, 3(1978), pp.417-439.

Sojka, Deborah. "Soft Selling Software" Datamation, 29, No.6(1983), pp.68-73.

Tharp, Alan L., and Woodrow E. Robbins. "Using Computers in Natural Language Mode for Elementary Education" International Journal of Man-Machine Studies, 7(1975), pp.703-725.

Thomas, John C. "Psychological Issues in Database Management" Third International Conference on Very Large Data Bases, Tokyo, Japan, 1977. New York: IEEE,1977, pp.169-185.

Thomas, John C., and John D. Gould. "A Psychological Study of QBE" Proceedings of the National Computer Conference 1975. Montvale,N.J.: AFIPS,1975, pp.439-445.

Tinnirello, Paul C. "Software Maintenance with Fourth Generation Languages" Proceedings of the National Computer Conference 1984. Montvale,N.J.: AFIPS,1984, pp.251-257.

Treu, Siegfried. "Interactive Command Language Design Based on Required Mental Work" International Journal of Man-Machine Studies, 7(1975), pp.135-149.

Truitt, Thomas D., and Stuart B. Mindlin. An Introduction to Nonprocedural Languages: Using NPL. New York: McGraw-Hill,1983.

Tyler, Micheal. "Cincom Shifts Gears" Datamation, 29, No.6(1983), pp.58-65.

Vessey, Iris. An Investigation of the Psychological Processes Underlying the Debugging of Computer Programs. Unpublished Doctoral Dissertation, University of Queensland, Australia, 1984.

Waldrop, James H. "Application Generators: A Case Study" Proceedings of the National Computer Conference 1982. Montvale,N.J.: AFIPS,1982, pp.365-368.

Wang, M.D. "The Role of Syntactic Complexity as a Determiner of Comprehensibility" Journal of Verbal Learning and Verbal Behavior, 9(1970), pp.398-404.

Weinberg, Gerald M., and Edward L. Schulman. "Goals and Performance in Computer Programming" Human Factors, 16(1974), pp.70-77.

Weissman, Larry. "Psychological Complexity of Computer Programs: An Experimental Methodology" SIGPLAN Notices, 9, No.6(1974), pp.25-36.

Welty, Charles, and David W. Stemple. "Human Factors Comparison of a Procedural and a Non-Procedural Query Language" ACM Transactions on Database Systems, 6(1981), pp.626-649.

Wilco, Elaine. "Systems Development Without Programming" Computer Data, 9, No.2(1984), p.19.

Youngs, Edward A. "Human Errors in Programming" International Journal of Man-Machine Studies, 6(1974), pp.361-376.

APPENDIX ONE

JUDGES' RATINGS OF THE SUBJECTS

RATING THE SUBJECTS

Please rate the subjects of the fourth generation language experiment on their experience with third generation languages. Use the information about the subjects given in the attached tables to rate the subjects.

The information supplied in the tables is structured as follows:

Column 1 The subject number.
Column 2 Educational degrees the subject has completed, or now has in progress.[1]
Column 3 The number of years the subject has been doing full-time work involving some degree of programming.
Column 4 The percentage of time spent programming (vs. doing other tasks) at work.
Column 5 The number of third generation languages the subject has used.
Column 6 The total number of third generation language programs the subject has written.
Column 7 The third generation language known best.
Column 8 The number of years experience the subject has with this language.
Column 9 The number of programs the subject has written in this language.
Column 10 to 12 Same as columns 7 to 9, but for the second best language known.
Column 13 to 15 Same as columns 7 to 9, but for the third best language known.

Rate the subjects experience with third generation languages using the following scale. Place your ratings on the attached RATING SHEET.

NOVICE EXPERT

1 2 3 4 5 6 7

---

[1]Note - Computer Systems is a two year diploma program offered at BCIT.

RATING SHEET

SUBJECT NUMBER	RATING
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	
26	
27	

SUBJECT NUMBER	RATING
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
51	
53	
55	
56	
57	
58	
59	
60	

SUBJECT NUMBER	DEGREES COMPLETED / IN PROGRESS	# OF YRS PROGRAMG AT WORK	%TIME PROG. WORK	# OF 3GLs KNOWN	TOTAL # PROGRAMS WRITTEN	LANGUAGES BEST KNOWN								
						LANG1	YRS	#PROGRAMS WRITTEN	LANG2	YRS	#PROGRAMS WRITTEN	LANG3	YRS	#PR WRT
1	COMP. SYS	-	-	6	25	PASCAL	2	6	BASIC	2	5	ASSEM	2	5
2	B.A., COMP. SYS	-	-	7	64	BASIC	3	30	ASSEM	3	11	C	1	10
3	B.MATH, M.SC.B.A.	-	-	1	6	FORTRA	1/2	6	-	-	-	-	-	-
4	B.Sc, COMP. SYS	-	-	6	54	PASCAL	4	30	FORTRA	3	8	COBOL	1	6
5	B.Sc, MBA	-	-	1	3	FORTRA	1	3	-	-	-	-	-	-
6	B.COMM, M.Sc.B.A.	-	-	2	55	FORTRA	1.5	40	COBOL	1	15	-	-	-
7	COMP. SYS	-	-	6	78	BASIC	6	30	PASCAL	1	20	ASSEM	1	12
8	COMP. SYS	.25	80	6	30	COBOL	1	15	PASCAL	1.5	5	ASSEM	1.5	5
9	COMP. SYS	-	-	5	41	PASCAL	4	25	BASIC	2	6	COBOL	1	5
10	B.B.A., COMP. SYS	-	-	4	35	BASIC	1/2	20	ASSEM	1/2	6	COBOL	1/2	6
11	B.COMM, MBA	-	-	2	9	COBOL	1/4	5	APL	1/4	4	-	-	-
12	COMP SYS	-	-	5	39	PASCAL	3	20	ASSEM	3	10	COBOL	2	5
13	B.Sc., MBA	-	-	2	5	APL	1/4	3	COBOL	1/8	2	-	-	-
14	COMP. SYS	-	-	6	30	PL/1	1	8	BASIC	1	5	COBOL	1	5
15	B. ENG (MECH)	3	5	3	33	FORTRA	1	20	BASIC	1	10	PASCAL	1/2	3
16	B.S.F., COMP. SYS	-	-	6	35	BASIC	1	10	PASCAL	1	6	COBOL	1	6
17	COMP. SYS	-	-	6	47	BASIC	6	30	COBOL	1	5	ASSEM	1	4
18	B.ENG, MBA	-	-	4	20	BASIC	1	10	FORTRA	1	5	COBOL	1/2	3
19	COMP. SYS	.5	25	6	51	PASCAL	2	30	FORTRA	2	8	ASSEM	1	5
20	B.Sc.	-	-	3	10	BASIC	1/4	6	PASCAL	1/4	3	COBOL	1/8	1
21	B.ENG, COMP. SYS	-	-	5	28	BASIC	1	10	FORTRA	1	10	ASSEM	1	6
22	B.COMM, MBA	-	-	2	13	BASIC	1/2	12	FORTRA	1/8	1	-	-	-
23	B.Sc., MBA	-	-	-	-	-	-	-	-	-	-	-	-	-
24	B.Sc., COMP. SYS	-	-	5	74	FORTRA	2	30	PASCAL	2	30	COBOL	1	7
25	B.ENG, MBA	-	-	1	7	FORTRA	2	7	-	-	-	-	-	-
26	COMP. SYS	-	-	5	10	ASSEM	1/2	4	COBOL	1/2	3	PL/1	1/2	2
27	COMP. SYS	-	-	7	39	PASCAL	4	12	COBOL	2	6	ASSEM	2	6
28	B.C.Sc., MBA	1	100	5	23	PASCAL	1	10	C	1	6	COBOL	1/2	3
29	B.COMM, MBA	-	-	3	23	FORTRA	5	20	APL	2	2	COBOL	1	1
30	B.A. (LINGUISTICS)	-	-	-	-	-	-	-	-	-	-	-	-	-
31	B.ENG., MBA	-	-	2	33	BASIC	2	30	FORTRA	1/2	3	-	-	-
32	COMP. SYS	-	-	6	27	BASIC	2	5	ASSEM	2	8	COBOL	1.5	5
33	COMP. SYS	-	-	5	38	COBOL	2	10	ASSEM	2	10	C	1	10
34	B.Sc., M.A.	8	5	3	32	FORTRA	10	20	APL	2	10	BASIC	1	2

SUBJECT NUMBER	DEGREES COMPLETED / IN PROGRESS	# OF YRS PROGRAMG AT WORK	%TIME PROG. WORK	# OF 3GLs KNOWN	TOTAL # LANGUAGES BEST KNOWN									
					PROGRAMS WRITTEN	LANG1	YRS	#PROGRAMS WRITTEN	LANG2	YRS	#PROGRAMS WRITTEN	LANG3	YRS	#PR WRT
35	COMP. SYS	-	-	5	87	ASSEM	4	50	COBOL	3	20	FORTRA	1/2	10
36	B.A., MBA	-	-	-	-	-	-	-	-	-	-	-	-	-
37	COMP. SYS	-	-	7	21	ASSEM	2	6	PASCAL	1	3	C	1	3
38	B.Sc, MBA	-	-	-	-	-	-	-	-	-	-	-	-	-
39	COMP. SYS	1/2	60	4	13	COBOL	2	6	ASSEM	2	4	PL/1	2	2
40	B.H.N., MBA	-	-	-	-	-	-	-	-	-	-	-	-	-
41	B.Sc.,M.Sc, MBA	-	-	2	5	APL	2	4	COBOL	2	1	-	-	-
42	COMP. SYS	-	-	5	37	ASSEM	1.5	13	COBOL	1	12	PASCAL	1/2	6
43	B.A. (ENGLISH)	-	-	-	-	-	-	-	-	-	-	-	-	-
44	B.ENG, M.A,PhD	-	-	5	46	BASIC	1/4	20	PASCAL	1	10	C	1/4	10
45	COMP. SYS	-	-	6	143	BASIC	7	50	PASCAL	6	30	COBOL	4	50
46	COMP. SYS	-	-	5	23	PASCAL	1/2	8	ASSEM	1/2	6	COBOL	1/2	5
47	B.H.E., MBA	-	-	-	-	-	-	-	-	-	-	-	-	-
48	B.A.(ECON), MBA	-	-	2	7	BASIC	4	5	APL	1	2	-	-	-
49	COMP. SYS	-	-	5	15	COBOL	2	6	ASSEM	1	4	PASCAL	1/2	2
51	COMP. SYS	-	-	5	11	ASSEM	1/2	4	COBOL	1/2	4	BASIC	1	1
53	COMP. SYS	-	-	5	27	ASSEM	2	10	COBOL	1.5	10	BASIC	2	3
55	COMP. SYS	-	-	5	24	BASIC	1	10	ASSEM	1	6	COBOL	1/2	4
56	COMP. SYS	-	-	7	320	BASIC	7	300	ASSEM	2	5	COBOL	1	5
57	COMP. SYS	-	-	6	40	BASIC	3	20	PASCAL	1/2	8	ASSEM	1	5
58	COMP. SYS	-	-	4	19	BASIC	2	6	COBOL	1	6	ASSEM	1	5
59	THEOLOGY, COMP. SYS	-	-	7	69	BASIC	1.5	40	ASSEM	1	12	PASCAL	1	7
60	B.A., COMP. SYS	-	-	6	29	ASSEM	2	10	FORTRA	1	6	BASIC	2	5

APPENDIX TWO

SIMPLE TEST

REPORT PREPARATION TEST

The database you will be using for the test problems is a university registration database. This database maintains information on which courses each student is taking, who teaches the courses, the student registration for each course and section, and personal data on the students. The database is composed of three physical files: 1) STUDSEG file (contains information on students) 2) PROFSEG file (contains information on the professors) and 3) REGISTER file (contains information on the courses and sections). Figure 1 illustrates the database, and figures 2 and 3 give the fields in each of the files.

Hierarchical Database for University Registration

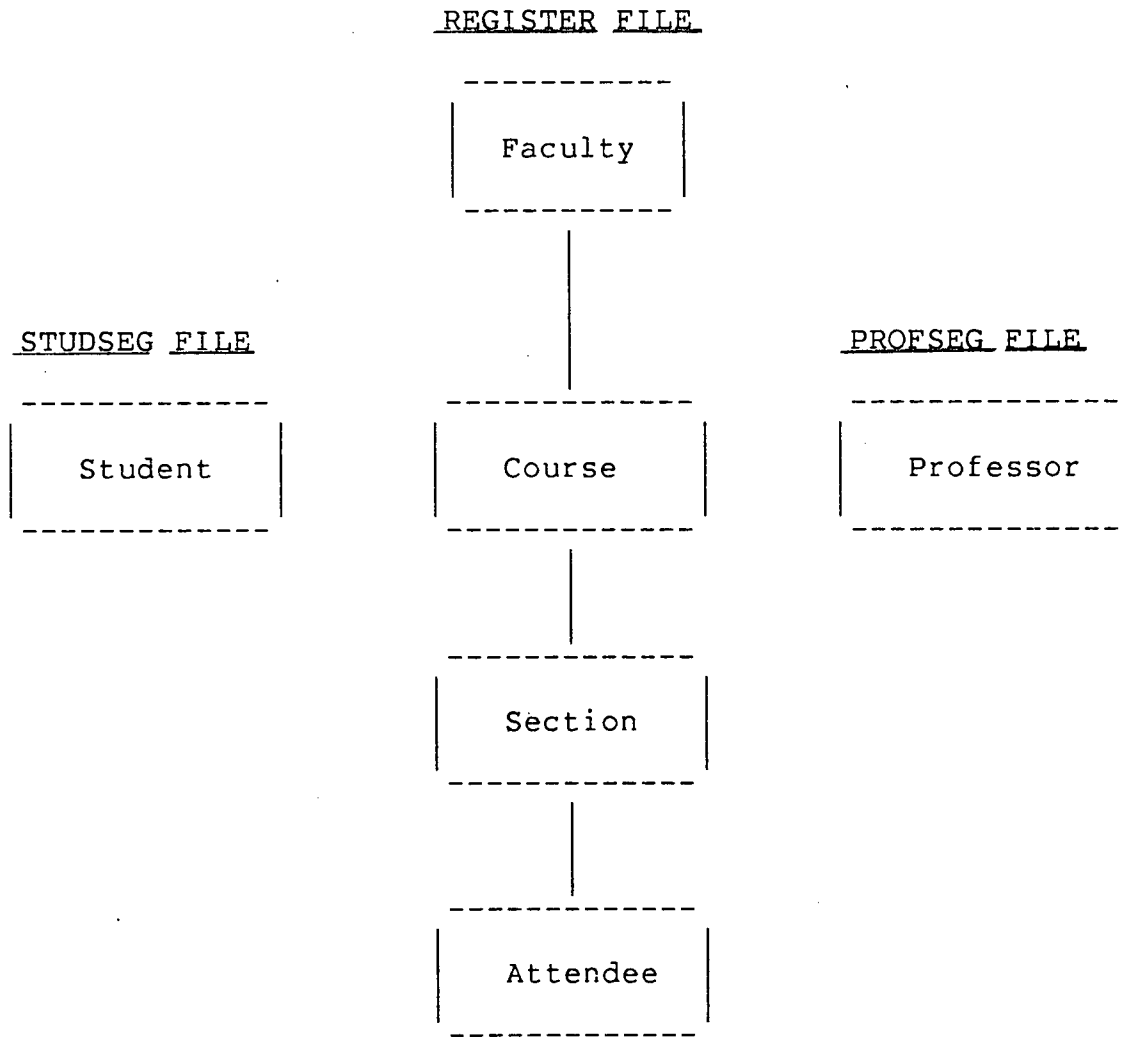


FIGURE 1

Description of Fields in the REGISTER File

1. FACULTY_NAME - This is the name of the faculty (Arts, Commerce etc.) in which the course is taught. This is an indexed field. ALIAS=FACNAME
2. COURSE_NUM - This is the course number (field of length 3 e.g 536). This is an indexed field. ALIAS=CONUM
3. COURSE_NAME - The name of the course, e.g. Accouting. ALIAS=CONAME.
4. CO_FEE - The dollar amount charged to the student for taking the course. ALIAS=COCHARG
5. SEC_NO - The section number of the course. Each course can have several sections (e.g 001, 002 etc.) ALIAS=SECNUM
6. PROF_TEACH - This is a numeric field of length 6 representing the identification number of the professor teaching the course. ALIAS=TEACH.
7. MAX_ENROLL - This is the maximum number of students which can be enrolled in the course section. ALIAS=MAX.
8. PERSON_ID - This is a numeric field of length 6 representing the identification number of the students enrolled in the course section. ALIAS=PERID.

Figure 2

Description of the Fields in the PROFSEG File

1. PROF_ID - This is a numeric field of length 6 representing the identification number of the professor. This is an indexed field. ALIAS=PROID.
2. PROF_NAME - The name of the professor. ALIAS=PROF.
3. OFFICE_NUM - The professor's office number. ALIAS=OFFNUM.
4. PR_FACULTY - The name of the faculty to which the professor belongs. ALIAS=PRFAC.

Description of the Fields in the STUDSEG File

1. STUDENT_ID - A numeric field of length 6 representing the identification number of the student. This is an indexed field. ALIAS=STUDID
2. STUDENT_NAME - The student's name. ALIAS=STUDNAME.
3. FACULTY - The name of the faculty to which the student belongs. ALIAS=FAC.
4. STREET_ADD - The student's home street address. ALIAS=STADDR
5. CITY - The student's home city. ALIAS=CT.
6. PROVINCE - The student's home province. ALIAS=PROV.
7. TEL_NUM - The student's home telephone number. ALIAS=TELNO.

Figure 3

Shown below are examples of data records in each of the three files.

STUDSEG FILE

STUDENT_ID	STUDENT_NAME	FACULTY	STREET_ADD	CITY	PROVINCE	TEL-NO
010101	JOE COOPER	COMMERCE	43 WAY ST.	PRINCETON	B.C	604-4343
693218	MILT JONES	ARTS	22 TREE ST.	GOLDEN	B.C	687-4396
⋮	⋮					
etc						

PROFSEG FILE

PROF_ID	PROF_NAME	OFFICE-NUM	PR-FACULTY
000173	DR. DOLLAR	21393	COMMERCE
⋮	⋮		
etc			

REGISTER FILE

FACULTY _NAME	COURSE NUM	COURSE NAME	CO FEE	SEC NO	PROF TEACH	MAX ENROLL	PERSON ID
COMMERCE	410	ACCOUNTING	299	001	000173	150	010101
"	"	"	"	002	000173	100	693218
ARTS	207	ENGLISH	99	001	000372	40	834449
"	"	"	"	"	"	"	673341
⋮	⋮						
etc.							

QUESTION 1

PART A

Show the structure of the report (column headings) produced by the following programs. Write the column headings exactly as FOCUS would produce them. If your second answer is the same as your first, just write "Same as above" in the space provided for the second problem.

```
TABLE FILE REGISTER
PRINT CO_FEE AND MAX
BY FACNAME BY CONUM BY SECNUM
END
```

```
TABLE FILE REGISTER
BY SECNUM
BY CONUM
BY FACNAME
PRINT CO_FEE MAX
END
```

PART B

Write a program which will display the course number, and section number and the number of students registered in the section, for every course and section in the database. Structure the report as follows.

COURSE_NUM	SEC_NO	PERSON_ID COUNT
.	.	.
	.	.
	.	.

PART C

Write a program which will produce the same fields as above but in the following matrix format.

	SEC-NO		
COURSE-NUM	001	002	003
107	25	40	5
250	.	.	.
333	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

NUMBER OF STUDENTS REGISTERED IN THE SECTION.

QUESTION 2

Write a program which will print the course name, and below it the course number and section number, for all the courses and sections in the database. Sequence the courses according to the faculty to which they belong. Show only one faculty per page. At the bottom of the report print "COURSE AND SECTION LIST AS OF JANUARY 31, 1986". Structure the report exactly as follows.

FACULTY_NAME

ARTS	COURSE_NAME	POLITICAL SCIENCE
	COURSE_NUM	199
	SEC_NO	001
	COURSE_NAME	POLITICAL SCIENCE
	COURSE_NUM	199
	SEC_NO	002

QUESTION 3

Write a program to produce a report which shows, for each section of the course names ENGLISH and FINANCE, 1) the ID number of the professor teaching, and 2) the maximum enrollment allowed in the course. Sequence the report by the faculty to which the course belongs, and by the course name and the section number, but do not print the section number. Also, show a subtotal for maximum enrollment each time the course name or faculty name changes. Produce the report and column headings exactly as follows:

FACULTY_NAME	COURSE_NAME	PROF_TEACH	MAXIMUM ENROLLMENT
* TOTAL COURSE_NAME	XXXXX		XXX
* TOTAL COURSE_NAME	XXXX		XXX
* TOTAL FACULTY_NAME	XXXXX		XXX

APPENDIX THREE

COMPLEX TEST

REPORT PREPARATION TEST

The database you will be using for the test problems is a university registration database. This database maintains information on which courses each student is taking, who teaches the courses, the student registration for each course and section, and personal data on the students. The database is composed of three physical files: 1) STUDSEG file (contains information on students) 2) PROFSEG file (contains information on the professors) and 3) REGISTER file (contains information on the courses and sections). Figure 1 illustrates the database, and figures 2 and 3 give the fields in each of the files.

Hierarchical Database for University Registration

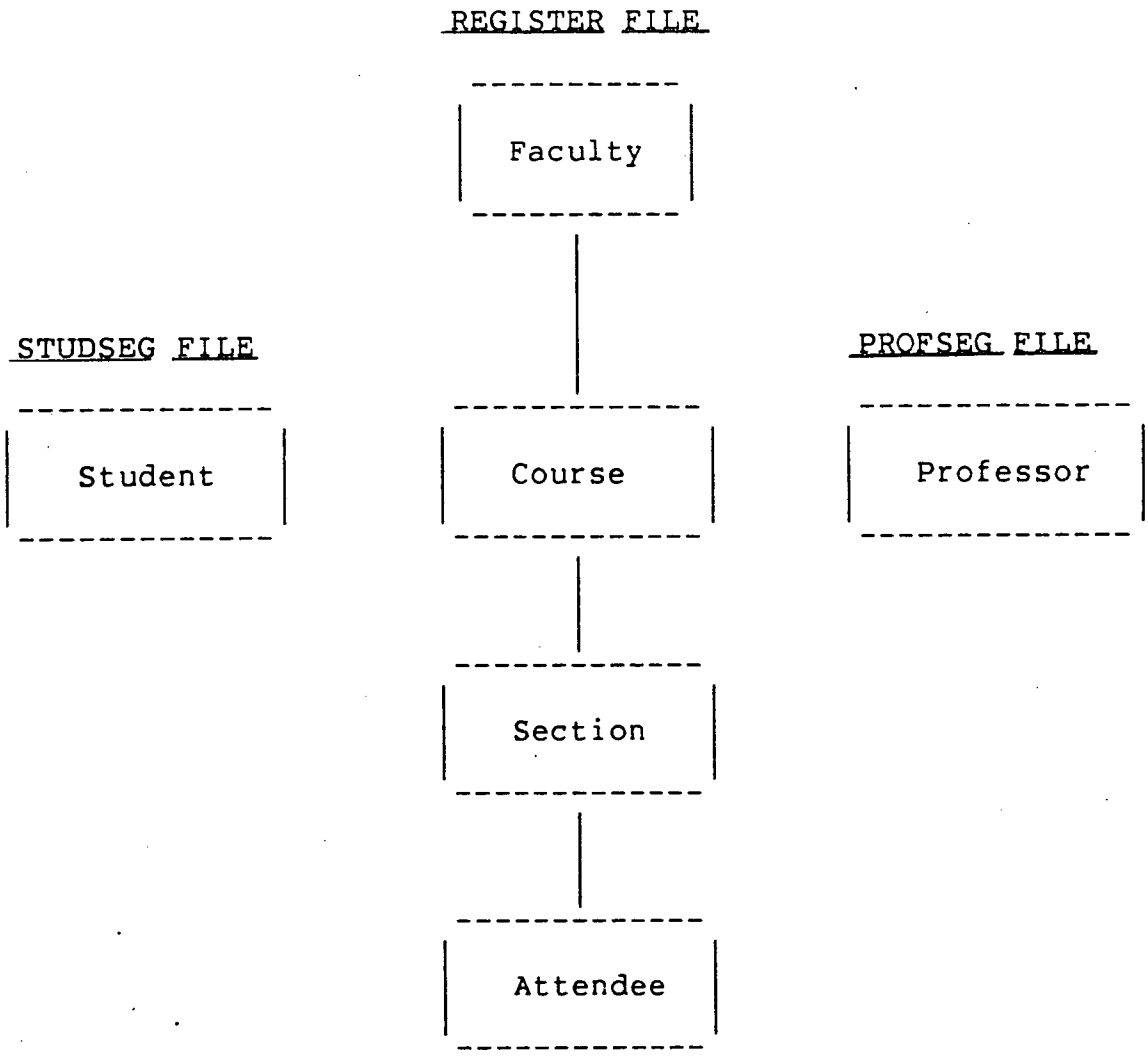


FIGURE 1

Description of Fields in the REGISTER File

1. FACULTY_NAME - This is the name of the faculty (Arts, Commerce etc.) in which the course is taught. This is an indexed field. ALIAS=FACNAME
2. COURSE_NUM - This is the course number (field of length 3 e.g 536). This is an indexed field. ALIAS=CONUM
3. COURSE_NAME - The name of the course, e.g. Accouting. ALIAS=CONAME.
4. CO_FEE - The dollar amount charged to the student for taking the course. ALIAS=COCHARG
5. SEC_NO - The section number of the course. Each course can have several sections (e.g 001, 002 etc.) ALIAS=SECNUM
6. PROF_TEACH - This is a numeric field of length 6 representing the identification number of the professor teaching the course. ALIAS=TEACH.
7. MAX_ENROLL - This is the maximum number of students which can be enrolled in the course section. ALIAS=MAX.
8. PERSON_ID - This is a numeric field of length 6 representing the identification number of the students enrolled in the course section. ALIAS=PERID.

Figure 2

Description of the Fields in the PROFSEG File

1. PROF_ID - This is a numeric field of length 6 representing the identification number of the professor. This is an indexed field. ALIAS=PROID.
2. PROF_NAME - The name of the professor. ALIAS=PROF.
3. OFFICE_NUM - The professor's office number. ALIAS=OFFNUM.
4. PR_FACULTY - The name of the faculty to which the professor belongs. ALIAS=PRFAC.

Description of the Fields in the STUDSEG File

1. STUDENT_ID - A numeric field of length 6 representing the identification number of the student. This is an indexed field. ALIAS=STUDID
2. STUDENT_NAME - The student's name. ALIAS=STUDNAME.
3. FACULTY - The name of the faculty to which the student belongs. ALIAS=FAC.
4. STREET_ADD - The student's home street address. ALIAS=STADDR
5. CITY - The student's home city. ALIAS=CT.
6. PROVINCE - The student's home province. ALIAS=PROV.
7. TEL_NUM - The student's home telephone number. ALIAS=TELNO.

Figure 3

Shown below are examples of data records in each of the three files.

STUDSEG FILE

STUDENT_ID	STUDENT_NAME	FACULTY	STREET_ADD	CITY	PROVINCE	TEL_NO
010101	JOE COOPER	COMMERCE	43 WAY ST.	PRINCETON	B.C	604-4343
693218	MILT JONES	ARTS	22 TREE ST.	GOLDEN	B.C	687-4396
⋮	⋮					
etc						

PROFSEG FILE

PROF_ID	PROF_NAME	OFFICE_NUM	PR_FACULTY
000173	DR. DOLLAR	21393	COMMERCE
⋮	⋮		
etc			

REGISTER FILE

FACULTY_NAME	COURSE NUM	COURSE NAME	CO FEE	SEC NO	PROF TEACH	MAX ENROLL	PERSON ID
COMMERCE	410	ACCOUNTING	299	001	000173	150	010101
"	"	"	"	002	000173	100	693218
ARTS	207	ENGLISH	99	001	000372	40	834449
"	"	"	"	"	"	"	673341
⋮	⋮						
etc.							

QUESTION 1

PART A

Show the structure of the report (column headings) produced by the following programs. Write the column headings exactly as FOCUS would produce them. If your second answer is the same as your first, just write "Same as above" in the space provided for the second problem.

```
TABLE FILE REGISTER
PRINT CO_FEE AND MAX
BY FACNAME BY CONUM BY SECNUM
END
```

```
TABLE FILE REGISTER
BY SECNUM
BY CONUM
BY FACNAME
PRINT CO_FEE MAX
END
```

PART B

Write a program which will display the course number, and section number and the number of students registered in the section, for every course and section in the database. Structure the report as follows.

COURSE_NUM	SEC_NO	PERSON_ID COUNT
.	.	.
	.	.
	.	.

PART C

Write a program which will produce the same fields as above but in the following matrix format.

COURSE-NUM	SEC-NO 001	002	003
107	25	40	5
250	.	.	.
333	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

NUMBER OF STUDENTS REGISTERED IN THE SECTION.

QUESTION 2

NOTE - For some of the following programs you may need to use the JOIN and DEFINE commands before issuing the TABLE command. Please show the commands exactly as you would have to type them on the computer.

Write a program which will print the course name, and below it the course number and section number together, in a field called NUMBER, for all the courses and sections in the database. Order the courses according to the faculty in which they belong. Structure the report exactly as follows:

```
FACULTY_NAME

ARTS            COURSE_NAME   POLITICAL SCIENCE
                NUMBER        199.001
                COURSE_NAME   POLITICAL SCIENCE
                NUMBER        199.002
                   .
                 etc.
```

QUESTION 3

Produce a course report for the Faculty of Arts. For each course, print the name of the course, the name of the professor teaching the section, the maximum enrollment allowed in the section, and the size of the classroom needed. Size of the classroom is defined as 'BIG' if the maximum enrollment is greater than 100, and 'SMALL' otherwise. Summarize each course by indicating the total maximum enrollment allowed (the sum of the maximums of the sections). Sequence the report by course number and section number, but do not print the course number with the rest of the fields. Instead, print the course number above the other course information as shown below. Your report should appear as follows (print column headings and summary lines exactly as they are shown below):

SEC_NO	COURSE_NAME	PROFESSOR	MAXIMUM ENROLLMENT	SIZE
COURSE NUMBER XXX				
001	POLI SCI	MR SMITH	60	SMALL
.	.	.	.	.
* TOTAL	COURSE_NUM XXX		XXX	
.				
.				
.				

APPENDIX FOUR

MARKING SCHEME FOR THE TESTS

MARKING SCHEME

General Guidelines

1. Unless otherwise noted the command lines between the first command, TABLE FILE, and the last command, END, can be placed in any order.

2. Do not subtract marks for mistakes which are obviously only spelling mistakes, but make sure they are only spelling mistakes not a reference to another data field or command.

3. Any fieldname can be replaced by its alias.

4. If commands are added which are not needed and

   i) which would cause the program to fail, subtract a 'noticeable' amount of marks.

   ii) which would unecessarily add to the program, but would not cause it to fail, subtract a 'minimal' amount of marks.

5. Commands which use the right keywords, but which are out of order, such as SUBTOTAL ON fieldname, or PRINT PERSON_ID COUNT, are wrong and should not receive any marks.

6. AND's are optional between fieldnames, and IS is equivalent to EQ.

QUESTION 1

PART A - 5 MARKS

The subject receives 1/2 mark for each column heading which is correctly placed, and has the correct heading. If either, the column is out of place, or the heading is incorrect, no marks are awarded.

SOLUTION

FACNAME CONUM SECNUM COFEE MAX

SECNUM CONUM FACNAME COFEE MAX

PART B -12 MARKS

SOLUTION	MARK
TABLE FILE REGISTER	1
COUNT PERSON_ID	4
BY COURSE_NUM BY SEC_NO	6
END	1

- If the verb is not COUNT for the field PERSON_ID, don't award any marks for that line.
- If the BY fields are reversed, i.e. BY SEC_NO BY COURSE_NUM, award only 3 of 6 marks.
- For each BY, or fieldname, which is missing, subtract 2 marks, up to a total of 6.
- If PRINT COURSE_NUM SEC_NO is used instead of BY COURSE_NUM BY SEC_NO, award only 2 of 6 marks.

PART C - 13 MARKS

SOLUTION	MARKS
TABLE FILE REGISTER	1
COUNT PERSON_ID	4
BY COURSE_NUM	3
ACROSS SEC_NO	4
END	1

- Lines 1 and 5 must be exactly as shown to receive marks.
- If the field is not PERSON_ID, or the verb is not COUNT, don't award any marks for line 2.
- If the field is not COURSE_NUM, or the command is not BY don't award any marks for line 3.
- If the ACROSS command is used, but with the wrong field, award 2 of the 4 marks.
- If 'BY SEC_NO' is used instead of 'ACROSS SEC_NO', award only 1 of the 4 marks.

QUESTION 2 - 25 MARKS

SOLUTION	MARKS
TABLE FILE REGISTER	1
PRINT COURSE_NAME OVER COURSE_NUM OVER SEC_NO	10
BY FACNAME	3
ON FACNAME PAGE-BREAK	5
FOOTING	4
"COURSE AND SECTION LIST AS OF JANUARY 31, 1986"	1
END	1

- Lines 1 and 7 must be exactly as shown to receive marks.
- If the OVER commands are left out of line 2, award only 3 of 10 marks.
- If the OVER fields are reversed i.e. 'OVER SEC_NO OVER COURSE_NUM' award 6 of 10 marks.
- If BY or ACROSS commands are used in place of OVER commands in line 2 , award only 3 of 10 marks.
- If a verb other than PRINT is used in the second line, subtract 4 marks. Also, subtract 2 marks for each fieldname which is missing.
- Line 3 must be exactly as shown to receive marks (but PAGE-BREAK could be added to line 3 instead of line 4).
- Award the 5 marks for line 4 if PAGE-BREAK is added to line 3 in place of line 4.
- Line 4 must appear exactly as shown to receive the marks. If the PAGE-BREAK is elsewhere than with the ON or BY command it is

incorrect.

- FOOTING can be on the same line as the quote.
- If SUBFOOT is used in place of FOOTING award 2 of 4 marks.
- If quotes are missing for line 6, do not award the mark.
- FOOTING CENTER is OK for line 5.

QUESTION 3 - 38 MARKS

SOLUTION	MARKS
TABLE FILE REGISTER	1
PRINT TEACH MAX_ENROLL AS 'MAXIMUM ENROLLMENT'	6
BY FACNAME BY CONAME BY SEC_NO	9
ON SEC_NO NOPRINT	5
ON CONAME SUB-TOTAL	10
IF CONAME EQ ENGLISH OR FINANCE	6
END	1

- Lines 1 and 7 must be exactly as shown.
- If line 2 is missing the AS command for the column title, award only 3 of the 6 marks.
- If TEACH is not in the PRINT line, but rather is in a BY field, this is acceptable (but it must be the last BY field).
- If the PRINT verb is replaced by SUM or COUNT, award only 3 of 6 marks.
- If one of the BY fields in line 3 is placed in a PRINT instead of a BY, subtract 3 marks for each mistake.
- If BY fields are out of order, award only 3 of 9 marks.
- Award 5 marks for line 4 as is, or if NOPRINT follows BY SEC_NO e.g. BY SEC_NO NOPRINT.
- Award 10 marks for line 5 as is, or if SUB-TOTAL follows the BY CONAME command e.g. BY CONAME SUB-TOTAL BY SEC_NO....
- If the ON field for line 5 is incorrect use your judgement as to how close the results would be to the one desired, don't award

more than 5 of the 10 marks.

- If SUBTOTAL is used in place of SUB-TOTAL don't award more than 5 of the 10 marks.

- If the ON field is wrong in line 5, and SUBTOTAL is used in place of SUB-TOTAL, don't award more than 3 of the 10 marks.

- If AND is used instead of OR in line 6 award 2 of the 6 marks.

- If line 6 is split into two lines as follows:

```
IF CONAME EQ ENGLISH
IF CONAME EQ FINAMCE
```

award 2 of the 6 marks.

- Naming the field to be subtotaled is acceptable, i.e. SUB-TOTAL MAX_ENROLL.

- If the following 2 lines replace line 5 subtract 1 mark

```
ON CONAME SUBTOTAL
ON FACNAME SUBTOTAL
```

COMPLEX TEST

QUESTION 2 - 26 MARKS

SOLUTION	MARKS
DEFINE FILE REGISTER	3
NUMBER = CONUM || '.' || SECNUM;	10
END	1
TABLE FILE REGISTER	1
PRINT CONAME OVER NUMBER	7
BY FACULTY	3
END	1

- The DEFINE commands must come before the TABLE commands. If they follow, award a maximum of 7 of the 14 marks (cut the marks in half).
- Lines 1,3,4,7 must be exactly as shown to receive the marks.
- If line 2 is placed inside the TABLE command rather than in a DEFINE, divide the marks for that line in half.
- If '|' is used instead of '||' in line 2, subtract 4 marks.
- If the '.' and one of the '||' are misssing, subtract 4 marks.
- If both of the above are wrong, award 4 of 10 marks.
- If the CONUM or SECNUM fields are replaced by other fields in line 2, award one mark each for '.', '||', '||', NUMBER and one of the correct fields.
- If a name other than NUMBER is used in the DEFINE, and the same name is also used in the TABLE, subtract 4 marks.
- If line 5 is correct, but NUMBER is not defined, award only 5

of the 7 marks.

- If the OVER command is missing in line 5, award only 3 of the 7 marks.

- Award 2 of the 7 marks if PRINT CONAME is correct, but the rest of the line is incorrect.

- Award 4 marks for PRINT CONAME OVER, if the OVER field is wrong.

- If the semi-colon is missing in the DEFINE, subtract 1 mark.

QUESTION 3 - 64 MARKS

SOLUTION	MARKS
JOIN TEACH IN REGISTER TO PROID IN PROFSEG AS NEW	10
DEFINE FILE REGISTER	3
SIZE = IF MAX_ENROLL GT 100 THEN 'BIG' ELSE 'SMALL';	10
END	1
TABLE FILE REGISTER	1
PRINT CONAME PROF AS 'PROFESSOR' MAX AS 'MAXIMUM ENROLLMENT' SIZE	9
BY CONUM BY SECNUM	6
ON CONUM NOPRINT	5
ON CONUM SUBTOTAL	6
SUBHEAD "COURSE NUMBER <CONUM>"	8
IF FACULTY EQ ARTS	4
END	1

- Lines 2,4,5,12 must be exactly as shown in order to receive the marks.
- If the files are reversed in the JOIN command, award 7 of the 10 marks, i.e. JOIN PROID IN PROFSEG TO TEACH IN REGISTER ...
- The name NEW in line 1 can be replaced by any other name.
- The name of the DEFINE field does not have to be SIZE, just as long as the field is later printed AS 'SIZE'.
- If quotes are missing in line 3, award 8 of the 10 marks.
- If the DEFINE block comes before the JOIN, award a maximum of 7 of the 14 marks.

- If line 3 is placed inside a TABLE block rather than in the DEFINE block, award a maximum of 4 of the 10 marks.
- If the DEFINE block follows the TABLE block, award a maximum of 5 of the 14 marks.
- If a COMPUTE, or IF statement is used in place of the DEFINE block, award a maximum of 4 marks.
- If the JOIN is not the first command, award only 5 of the 10 marks.
- CONAME, SECNUM, and PROF can be either BY fields, or fields in the PRINT command. But, they must be in the proper order
- Subtract 2 marks for each AS command which is executed incorrectly or is missing.
- Subtract 2 marks for each PRINT or BY field which is missing, up to a maximum of 12 marks. If the BY CONUM field is missing subtract 4 rather than 2 marks.
- Award a maximum of 6 marks for lines 8 and 9 together if BY CONUM is missing.
- NOPRINT must be associated with the CONUM field in order to receive the 5 marks.
- SUB-TOTAL can be used in place of SUBTOTAL.
- SUBTOTAL MAX_ENROLL is acceptable.
- If SUMMARIZE is used in place of SUBTOTAL, award only 4 of 6 marks.
- If the SUBTOTAL field is not CONUM don't award any marks.
- In line 10, award 3 marks for SUBHEAD, 3 for <CONUM>, and 2 for the text "COURSE NUMBER".

APPENDIX FIVE

EXPERIMENTAL PROCEDURES

EXPERIMENTAL PROCEDURES

1) Brief Introduction of the experiment

- Brief explanation of the purpose of the experiment (comparing the ability of novice and experienced programmers to learn a 4GL)
- Stress how important 4GL's could be to the subjects in the future.
- Stress that the session is also important because it supplies the experimenter with data for his thesis.
- Subjects should sit at least one seat apart if possible.

2) Explain the sequence of events for the session

- The session will take 3 hours, maybe more . Stress to the subjects that they have to run through all the experiment (reading, sample problems, test, questionnaire), otherwise the data will not be of value to the study. At this point they should decide whether they want to commit to the three hours involved.
-Stress that there should be no talking at any time during the experiment. If the subjects have problems they should consult one of the lab assistants.
- Tell the subjects that the first step involves reading the manual, which will introduce them to some basic FOCUS reporting commands. Inform them that they can keep the manual for the problem session, and for the test which follow.
- Tell the subjects they should read the manual carefully because it will make the sample problems and test much easier.
- Stress to them that there is no time limit on either the

reading, or sample problem sessions.

- When the subject is finished reading, he should get up quietly, and inform the lab assistant that he has finished. They should not disturb the other subjects. - Explain that the next step after reading, is a practice session on the computer. Eight practice problems have to be completed. Solutions are provided, but the subjects should try to solve the problem at least once before looking at the solutions. Try to budget about an hour, or 8 minutes per problem for the sample problems. When the subjects are finished they should inform the lab assistant, who will give them the test. Subjects will be allowed 45 minutes for the test. After completing the test the subjects will be asked to fill out a questionnaire on their computer background. This will give the experimenter information concerning their level of programming experience.

3) What the lab assistant should do during the experiment

- Hand out the FOCUS manuals to subjects.
- Record the reading starting time for each subject.
- Lab assistant will record the time the subject finished reading the manual.
- Give the subject the set of sample problems, when he has finished reading the manual.
- Record the time the subject starts the sample problems.
- Record the time when the subject has finished the sample problems.
- Give the subject the appropriate test (either complex or

simple), and tell him he has 45 minutes.

- Record the time the subject starts the test.
- Advise the subject when he has only two minutes left.
- Record the subject's finish time.
- Make sure the subject's name is on the test.
- Give the subject the questionnaire, and collect it when he has finished. Check to make sure the subject has filled out all the the questionnaire.

APPENDIX SIX

REPORT GENERATION TRAINING MANUAL

FOURTH GENERATION LANGUAGES

REPORT GENERATION TRAINING MANUAL

(Adapted from PC/FOCUS User's Manual)

C. Pulfer

Report Generation

Before beginning the report generation course it is important that you understand some simple database concepts. Three important terms which you should know are: *files, records, and fields* . We can think of the computer as being equivalent to a filing cabinet. Within the filing cabinet we might have an employee file, containing information on all of the company's employees. Similarly we can have an equivalent employee file on the computer, in a database. A *database* might contain many of these types of files. For example, the computer might have a file on employees and also a file on shareholders, within the database. In our filing cabinet file on employees we have information on many employees. The information concerning only one employee is called a *record*. Therefore we have as many records as we have employees. We would have a record for employee John Smith, one for Doug Johnson, one for Don Wilson etc. The record contains information on just one employee. Each piece of information on the employee is called a *field*. For example we might have a salary field listing the salary of the employee, and an experience field listing how many years of experience the employee has.

TABLE OF CONTENTS

Introduction

This Report Preparation Manual will introduce you to a computer package called FOCUS. FOCUS is a database management system which allows users to store, maintain, and report on data which is of interest to them. FOCUS is a "fourth generation", or high productivity language which has gained wide acceptance in the business community. What you learn today should be useful to you in the future.

Typically, businesses will want to store data on their personnel, the firm's financial position etc. Decision-makers, within the business, will want to see regular reports on how things are progressing in recruiting, sales etc. The report preparation commands within FOCUS can be used to prepare these reports. The rest of the manual will introduce you to these FOCUS commands.

Most of the FOCUS commands are relatively English-like and easy to understand, though some are more difficult than others. Read the manual and learn the commands carefully. There is no time limit for reading the manual. Take as much time as you feel is necessary. Once you are finished, you will be tested on the commands you have learned. Notify the lab assistant once you are finished.

REPORT WRITING

FOCUS can be used to enter data, maintain data and report on data. Issuing the command TABLE FILE filename, where filename is the name of the file containing the information which will appear in the report, allows us to enter the reporting mode. The TABLE FILE command must be the first command in the report program. For example, if we have a file containing employee names and salaries, called SALARY, which we want to use in our report, we would issue the command TABLE FILE SALARY.

```
TABLE FILE SALARY
```

Other report commands would then be issued, followed finally, by the word END, on a line by itself.

```
TABLE FILE SALARY
      .
      .
      .
      .
END
```

a) Request Statements

"Request statements" are the commands which are used to produce the reports a user desires. A report request statement follows the rules of an imperative English sentence. The sentence begins with a verb of action which is followed by verb objects, then a series of phrases.

The examples which follow all use a sample file called PRODUCT which contains the following fields:

PRODUCT-TYPE	Identity of product
AREA	Geographical area
CUSTOMER	Name of Customer
MONTH	Month from 1 to 12
UNITS	Number of units shipped
AMOUNT	Dollar value of shipment

A field can also be refered to by its ALIAS. The ALIAS is a shorter version of the fieldname, making it easier for

the user to type in his request statements. If the PRODUCT-TYPE field has as its ALIAS, PROD, the user could issue either of the following equivalent commands:

i) PRINT PRODUCT-TYPE
ii) PRINT PROD

The order of presentation of the command elements within a report is arbitrary. The following are all equivalent commands:

i) SUM UNITS
BY MONTH

ii) SUM UNITS BY MONTH

iii) BY MONTH SUM UNITS

1. VERBS

A verb is a word of action. The action is performed on the fields which are named as the objects of the verb. The list of verbs are:

Verb	Meaning
PRINT	List the fields desired, with a different record on each line
COUNT	Count the number of occurences of a field, and display the results.
SUM	Add the numeric fields of the records together, and display the results.

The syntax (structure of the command) for a simple verb phrase is:

VERB fieldname [AND] fieldname [AND] fieldname etc
(the AND between fieldnames is optional)

Examples
```
SUM AMOUNT
PRINT PRODUCT-TYPE AND AREA
COUNT PRODUCT-TYPE
```

a) PRINT

The PRINT command causes the information in the fields desired to be listed. The order in which fieldnames are provided in the verb phrase is the order in which the columns of the report are printed.

*NOTE - The PRINT verb cannot be used in the same program as the SUM or COUNT verb. This is because PRINT, and SUM or COUNT, display different amounts of incompatible information. PRINT does not involve any summarization of information contained in the database, as COUNT and SUM do. PRINT simply lists the information which exists in the database.

Example
```
TABLE FILE PRODUCT
PRINT PRODUCT-TYPE UNITS
END
```

This produces the following report containing the fields PRODUCT-TYPE and UNITS extracted from each record in the file PRODUCT.

PRODUCT-TYPE	UNITS
AXLES	150
BEARING	324
.	.
.	.
.	.

b) SUM

When the total of the values of the data field (for all records in the file) is required then the verb SUM is used. If the PRODUCT-TYPE and UNITS fields look like this

PRODUCT-TYPE	UNITS
AXLE	150
BEARING	324
SCREW	300
BOLT	90
.	.
.	.

SUM UNITS would result in only one piece of information being displayed, i.e. the total of the UNITS field.

UNITS
1545

We can also produce SUMS for portions of the file by using the BY command. For instance, SUM UNITS BY MONTH means that all of the values of the field UNITS are to be added together for each MONTH. In this example there will only be 12 lines on the printed report, one for each month.

The report appears as:

MONTH	UNITS
1	18000
2	14625
3	10843
.	.
.	.

Example 2

```
TABLE FILE PRODUCT
SUM UNITS AND AMOUNT
BY AREA
END
```

These commands produce the following report

AREA	UNITS	AMOUNT
EAST	10000	26000
NORTH	8000	19000
.	.	.
.	.	.

Note - The SUM command cannot be used to show column-totals, for PRINTed fields, at the bottom of a report. As we will see, COLUMN-TOTAL or SUBTOTAL can be used for this purpose.

Sorting With the BY Command

A phrase in the request statement beginning with the word BY means to sequence the lines of the report by the field whose name follows (see above example). Multiple BY commands can be used. The first BY field specified would be the major sort field, the second BY sorts within each occurence of the first By field etc. For example the following report has three By fields:

```
TABLE FILE PRODUCT
SUM UNITS
BY AREA BY MONTH BY PRODUCT-TYPE
IF MONTH IS 1 OR 2 OR 3
END
```

The report appears as:

AREA	MONTH	PRODUCT-TYPE	UNITS
EAST	1	BOLTS	200
		FLANGES	125
	2	BOLTS	600
		FLANGES	800
	3	BOLTS	625
		FLANGES	515
NORTH	1	BOLTS	125
		FLANGES	315
		.	
		.	
		etc.	

The default sequence of ascending (LOW TO HIGH) sort, is used to present the report, (alphabetically A-Z and numerically 1,2,3...). This can be changed to a descending sequence, Z-A and highest numbers first, by the command BY HIGHEST fieldname.

c) COUNT

A count of the number of occurences of some data field. (i.e. in how many records does the field occur, in the database)

Example

```
COUNT CUSTOMER
BY AREA
```

This produces:

AREA	CUSTOMER COUNT
EAST	248
NORTH	172
.	
.	

SUM and COUNT can be combined in one command.

```
SUM AMOUNT AND COUNT CUSTOMER
BY AREA
```

This report would produce the same report as the example above, except that another column AMOUNT would be added.

Producing a Matrix Report

Producing a matrix display is accomplished by combining the BY command with an ACROSS command. In this case the columns are spread ACROSS some variable of interest in addition to the sort fields controlling the rows of the report. Note the use of the phrase ACROSS MONTH in the following example, and the command 'COLUMN-TOTAL' which produces column totals.

```
TABLE FILE PRODUCT
COUNT PRODUCT-TYPE AND COLUMN-TOTAL  BY REGION
ACROSS MONTH
END
```

	MONTH								
REGION	1	2	3	4	5	6	7	8	9
EAST	20	10	15	16	19	.	.	.	.
NORTH	14	70	21	26	28				
SOUTH	18	10	14	19	19	.	.	.	.
WEST	15	8	7	4	9	.	.	.	.
TOTAL	68	48	57	65	75	.	.	.	.

The values in the matrix represent the number of products available in the region by month.

The number of columns displayed on the report is equal to the number of verb object fields times the number of values retrieved for the ACROSS field. For instance if the phrase ACROSS MONTH is used and there are two verb objects (PRINT AMOUNT and UNITS), then there will be 24 output columns, composed of 12 pairs.

The report title appears as

```
MONTH
1                    2                    3
AMOUNT     UNITS     AMOUNT     UNITS     AMOUNT     UNITS ....
-------------------------------------------------------------
```

2. Displaying Data Fields Over Each Other

Normally one column is occupied by one data field and its title heads the column. This can be reversed by specifying that the data fields are to appear one over the other. Instead of connecting the verb object fields with the word AND, the word OVER is used. For example:

```
SUM AMOUNT OVER UNITS
BY AREA
```

This would produce:

AREA		
EAST	AMOUNT	4050
	UNITS	487
NORTH	AMOUNT	2686
	UNITS	456

3. AS

The default column-title (which is the filedname) can be replaced by a more meaningful title by the use of the phrase "AS column-title".

Example

```
PRINT PCT.AMOUNT AS 'PERCENTAGE OF AMOUNT'
```

would produce the column title PERCENTAGE OF AMOUNT instead of PCT AMOUNT

4. Grouping Numerical Data

Care must be taken when sorting by numeric fields. Each different value (perhaps only different in the last decimal place) would result in a separate line or column. A facililty is provided to sort data in ranges of values. A convenient way to view numerical data is to group the values into ranges and display the results in these ranges.The command IN-GROUPS-OF can be used to group numerical values in desired groups.

e.g.
```
TABLE FILE PRODUCT
COUNT PRODUCT-TYPE
BY UNITS IN-GROUPS-OF 500
END
```

Produces

UNITS	PRODUCT-TYPE COUNT
0	40
500	85
1000	72
1500	58
2000	14
2500	8

5. Record Selection

In general any command accesses all records in the file. If we only want to access certain fields, we can use the IF command. Any number of IF phrases can be used, and they may refer to any data field in the file. The syntax is

IF fieldname RELATION literal [OR literal OR literal]

where RELATION are operators as shown below, and
where LITERAL is either a numeric constant (e.g. 34) or a character constant (e.g. 'STEEL')

e.g. IF AREA IS EAST

Here 'IS' is the RELATION and 'EAST' is the LITERAL

Only those records which contain the value EAST in the field AREA are accessed, the others are ignored.

RELATION	MEANING

Note - Relations separated by commas are equivalent

IS,EQ	Equality between field value and literal
IS-NOT, NE	Inequality between field value and literal
IS-FROM, GE	Field value equal to or greater than literal
TO, LE	Field value equal to or less than literal
EXCEEDS, GT	Field value greater than literal
IS-LESS-THAN, LT	Field value less than literal
FROM TO	Field value in range
NOT-FROM TO	Field value not in range
CONTAINS	Characters in field value contains chracter in literal
OMITS	Characters in a field value do not contain characters in literal. Opposite of CONTAINS

Examples

IF AREA IS EAST OR WEST

Here, only those records which contain the value EAST, or WEST in the field AREA are accessed, the others are ignored.

Character fields such as EAST do not have to be enclosed in single quotes unless the field contains two or more words separated by blanks such as 'NEW YORK'.

IF AMOUNT IS-FROM 100
The field AMOUNT must be equal to or greater than the value 100

IF PRODUCT-TYPE CONTAINS STEEL
The field PRODUCT-TYPE must contain the characters "STEEL" anywhere within it e.g. COLDSTEEL.

IF AMOUNT EXCEEDS 40
IF UNITS FROM 100 TO 140
The field UNITS must have a value between 100 and 140 inclusive.

IF UNITS NOT-FROM (4 TO 6) OR (9 TO 11)
The field value must be equal to a number outside the ranges given.

NOTE - Two IF conditions cannot be placed on the same line. For instance IF AMOUNT EXCEEDS 40 AND UNITS EQ 100 is incorrect. The two conditions should be placed on separate lines as follows: IF AMOUNT EXCEEDS 40
IF UNITS EQ 100

6. Control Conditions

Within FOCUS, a variety of actions are provided which pertain to what happens on the printed report when a sort control field (BY) changes value. For instance, a sub-total may be displayed. The syntax for specifying the control conditions uses a phrase beginning with the word 'ON'. This is followed by the name of the field. When this field changes value on the printed report, then just before the next value is printed, the action mentioned is taken.

ON FIELD-NAME

SUB-TOTAL	accumulate and display all sub-totals
SUBTOTAL	accumulate and display single subtotal
PAGE-BREAK	force a page break
SKIP-LINE	insert a blank line
FOLD-LINE	fold a long line
SUMMARIZE	summarize calculations at all sort breaks.
RECOMPUTE	summarize only at named sort breaks
NOPRINT	suppress printing of this column
UNDER-LINE	draw underline across page
SUBFOOT	insert free text after values
SUBHEAD	insert free text before values
COMPUTE } RECAP	compute recaps of content line

a) SUB-TOTAL

Under each column of numeric data, a subtotal is printed. Each sub-total line displays the running accumulations of the sort field up to its last break in value. A COMPLETE GRAND TOTAL FOR EACH COLUMN IS PRODUCED AUTOMATICALLY.

All of the sub-totals are displayed up to and including the point of sort break requested so that only the inner-most point of sub-totalling should be requested. For instance, if the 'BY' fields are

```
BY AREA
BY PRODUCT-TYPE
BY MONTH
ON PRODUCT-TYPE SUB-TOTAL
```

Then when the AREA changes SUBTOTALS FOR BOTH PRODUCT_TYPE AND THE INNER SORT FIELD AREA ARE DISPLAYED.For example

```
TABLE FILE PRODUCT
PRINT UNITS AND AMOUNT
BY AREA BY PRODUCT-TYPE BY MONTH FROM 1 TO 3
ON PRODUCT-TYPE SUB-TOTAL
END
```

AREA	PRODUCT-TYPE	MONTH	UNITS	AMOUNT
EAST	BEARINGS	1	100	50.45
		2	140	61.75
		3	210	76.49
*TOTAL	PRODUCT-TYPE BEARINGS		450	188.69
	FLANGES	1	125	64.40
		2	115	91.38
		3	143	63.51
*TOTAL	PRODUCT-TYPE FLANGES		383	219.29
*TOTAL	AREA EAST		833	407.98
WEST	AXLES	1	100	130.50

.
.
.
etc.

b) SUBTOTAL

When the word SUBTOTAL without a hyphen is used then only the subtotal of the sort break field, mentioned in the 'ON' phrase, is displayed. The inner 'BY' fields ARE NOT DISPLAYED . For instance:

```
BY AREA
BY PRODUCT-TYPE
BY MONTH
ON PRODUCT-TYPE SUBTOTAL
```

This will display a subtotal when the PRODUCT-TYPE field changes value, but WILL NOT display a subtotal for the outer field AREA when it changes value. For example

```
TABLE FILE PRODUCT
PRINT UNITS AND AMOUNT
BY AREA BY PRODUCT-TYPE BY MONTH FROM 1 TO 3
ON PRODUCT-TYPE SUBTOTAL
END
```

AREA	PRODUCT-TYPE	MONTH	UNITS	AMOUNT
EAST	BEARINGS	1	100	50.45
		2	140	61.75
		3	210	76.49
*TOTAL PRODUCT-TYPE BEARINGS			450	188.69
	FLANGES	1	125	64.40
		2	115	91.38
		3	143	63.51
*TOTAL PRODUCT-TYPE FLANGES			383	219.29
WEST	AXLES	1	100	130.50

.
.
.
etc.

SUBTOTAL for Specific Fields

A list of specific fields to subtotal can be supplied after the word SUB-TOTAL, or SUBTOTAL, is typed. This list overrides the default which includes all numerical verb object fields.
For example

```
TABLE FILE PRODUCT
SUM UNITS AND AMOUNTS
BY PRODUCT-TYPE BY MONTH
ON PRODUCT-TYPE SUBTOTAL UNITS
END
```

Here only the UNITS field will be subtotaled (normally UNITS and AMOUNT would be subtotaled).

The syntax is

ON by field { SUB-TOTAL / SUBTOTAL } fieldname AND fieldname ...

Combined PAGE-BREAK and SUB-TOTAL

The general use of a page break is to provide a separate report for each major sort control value. If a sub-total is also requested then each of the separate reports may or may not have the running subtotals accumulated at the same time the page break is forced.

The rule which is followed is:

- If the page break is requested first, then each such break will be given only the sub-totals which pertain to it, and will not accumulate running totals into future pages.

- If the sub-total is requested first, followed by the page break, then it is assumed that the standard way of producing sub-totals is desired and that the subtotals will accumulate into subsequent subtotals on other pages.

c) SKIP-LINE

The SKIP-LINE command can be used when referring to sort control (BY) fields or to verb objects.

The following example produces a blank line each time the area changes.

```
BY PRODUCT-TYPE BY AREA
ON AREA SKIP-LINE
```

The next example creates a doubled spaced report: a line is skipped after each PRODUCT-TYPE

```
PRINT PRODUCT-TYPE
ON PRODUCT-TYPE SKIP-LINE
```

d) SUMMARIZE

Summarization is similar to a sub-total. The difference depends upon whether any columns in the output report are themselves the result of a calculation or a calculated field. If they are, then the indicated calculation is performed instead of a sub-total on the summary line of the report.

Example

```
TABLE FILE PRODUCT
SUM AMOUNT AND UNITS
COMPUTE
PER UNIT = AMOUNT/UNITS;
BY PRODUCT-TYPE BY AREA
ON PRODUCT-TYPE SUMMARIZE
END
```

PRODUCT-TYPE	AREA	AMOUNT	UNITS	PER UNIT
AXLES	EAST	1,342.50	1500	.89
	NORTH	2,761.41	2000	1.38
	SOUTH	3,849.52	3700	1.04
	WEST	2,147.36	1800	1.17
*TOTAL AXLES		10,100.79	9000	1.12

The COMPUTE verb is used to produce the new field PER UNIT. Compute can be used to produce new fields resulting from the normal math operators: / (divide), * (multiply), - (subtract), + (add). Note that the PER UNIT field is displayed automatically, no PRINT command is needed. The COMPUTE verb acts like a PRINT verb in this case. The COMPUTE verb can never be the first command of a program.

Notice that in the column titled PER UNITS the value printed is 1.12. This is 10,100.79 / 9000, and is the result of the same calculation as the other numbers in the column. Had a sub-total been requested, the sum of the numbers in the PER UNIT column would have appeared. This would have been 4.50 and would have been meaningless in this situation.

e) RECOMPUTE

Exactly like summarize except only the values for the specific 'BY' field requested are displayed, not the higher level 'BY' fields. The difference between SUMMARIZE and RECOMPUTE is equivalent to the difference between SUB-TOTAL and SUBTOTAL.

f) NOPRINT

This is used when we do not want one of the sort control fields to be printed in the final report. The report will still be sequenced by this field, but both the column

title and the data for that column are removed from the final printed report.

```
e.g.    BY AREA
        BY MONTH
        ON AREA NOPRINT
```

The AREA field will not be printed in the report.

g) COMPUTE - RECAP LINES

The facility to produce calculations in addition to or instead of sub-totals at desired breaks in the sort control fields is provided as a recap line. A recap line is a line which is calculated, based on the data in the content lines, every time a control field changes value. In a way a sub-total is a recap line, but because it is so common it can be directly requested. Instead of just totalling the fields, as is done for subtotals, RECAP can be used to display averages or ratios at the sort break.

A recap is performed whenever the control field changes value.

```
ON fieldname {COMPUTE}
             {RECAP  }
```

COMPUTE and RECAP are equivalent when used with 'ON'

This is followed by the calculations to be performed.

```
e.g     TABLE FILE PRODUCT
        SUM AMOUNT AND UNITS AND COUNT
        BY AREA BY MONTH FROM 1 TO 3
        ON AREA RECAP
        UNITPRICE = AMOUNT/UNITS;
        AVE SHIPMENT = UNITS/COUNT;
        END
```

AREA	MONTH	AMOUNT	UNITS	COUNT
EAST	1	4528	200	8
	2	1200	110	13
	3	6240	460	14
*EAST		UNITPRICE	15.55	
		AVE SHIPMENT	30.80	
NORTH	1	1400	200	7
	.			
	.			
	etc.			

h) HEADING and FOOTING

A report heading can be supplied by giving the command HEADING followed by the title enclosed in double quotes. For example

```
HEADING
"PRODUCT REPORT"
"AS OF DEC 31, 1986"
```

would produce

```
PRODUCT REPORT
AS OF DEC 31, 1986
```

at the top of the report.

To center the heading use the command HEADING CENTER followed by text in quotes.

To produce text at the end of the report, we would use the same syntax as for HEADING, but we would use the command FOOTING.

i) UNDER-LINE

This command is used to draw an underline after the named field changes. The line is drawn after any other option such as RECAP or SUB-TOTAL.

j) SUBHEAD

This command is used to insert text before a control break caused by a BY command. When data is to be embedded in

text the fieldnames are enclosed in brackets i.e. "text <fieldname> text". The data values are those which would appear on the first line of the control break had this data been placed on the control break line rather than in the SUBHEAD.

e.g.
```
TABLE FILE PRODUCT
SUM UNITS AND AMOUNT
BY AREA BY PRODUCT-TYPE
ON AREA NOPRINT AND SUBHEAD
" SUMMARY FOR <AREA>"
END
```

PRODUCT-TYPE	UNITS	AMOUNT
SUMMARY FOR EAST		
AXLES	650	141
BEARINGS	720	182
SUMMARY FOR WEST		
AXLES	534	162.45
.		
.		
etc.		

Two special field prefixes are applicable to SUBHEADS

<ST.fieldname	The subtotal value of the field at that point in the report.
<CT.fieldname	The running column total of the field at that point in the report.

k) SUBFOOT

This command is equivalent to SUBHEAD, but the text appears after, rather than before, the control break.

7. DEFINE command

The DEFINE command is used to create temporary data fields. Temporary data fields can be defined as mathematical or logical combinations of real or other temporary data fields. These temporary fields can be used in report request statements.

Some of the uses of temporary data fields are (don't worry if you don't understand these right away):

- Compute new numerical values which are not in a data record

- Compute conditional numerical values based on IF-THEN-ELSE conditional logic

- Compute new strings of alphanumeric characters from other strings.

DEFINE syntax

```
DEFINE FILE filename
```

where filename is the name of the file you are using to produce a new field.

```
name = expression;      name can be up to 12
                        characters long
name = expression;
       .
       .
       .
END
```

expression - is the calculation, or conditional calculation to be performed. It must be terminated by a semi-colon.

In the following example the file PRODUCT contains the fields AMOUNT and UNITS and will contain the new field PRICE. The field PRICE could then be used in a report program begun with TABLE FILE PRODUCT. But, DEFINE must take place before using the new field in a TABLE command.

e.g.
```
DEFINE FILE PRODUCT
PRICE = AMOUNT/UNITS; (NOTE THE USE OF THE
END                    SEMICOLON)
```

The arithmetic operations available are

```
+  plus
-  minus
*  multiply
/  divide
** exponentiation
```

The logical operators are

```
EQ   equal
NE   not equal
LE   less than or equal
LT   less than
GE   greater than or equal
GT   greater than
AND  logical connective AND
OR   logical connective OR
```

These operands are used most frequently in conditional calculations of the IF-THEN-ELSE type.

```
e.g.    DEFINE FILE PRODUCT
        NEW-VAL = IF AMOUNT LT 100 THEN AMOUNT*1.5
                  ELSE AMOUNT*2.0;
        END
```

This sets NEW-VAL to either AMOUNT*1.5 or AMOUNT*2 depending on the value of AMOUNT

```
TFACTOR = IF AMOUNT GT 100 OR PRICE LT FACTOR
          THEN AMOUNT ELSE AMOUNT * FACTOR;

SIZE = IF UNITS GT 100 THEN 'LARGE' ELSE 'SMALL';
```

Here a character string (either LARGE or SMALL) is assigned to the field SIZE.

```
TYPE = IF PRODUCT EQ 'AUTO' THEN 1 ELSE 0;
```

1. NOTE ALPHANUMERIC LITERALS MUST BE ENCLOSED IN SINGLE QUOTES

2. EACH EXPRESSION MUST END IN A SEMI-COLON

a) Concatenating Character Strings

Concatenation is used to join fields. Two or more character strings of alphanumeric fields and/or constants can be combined into a single field. In this way, the values of different fields can be joined in a new field.

e.g.
```
DEFINE FILE PRODUCT
FULLNAME = PLANT | AREA | '3'
END

if PLANT = GEOR in a 6 character field
   AREA = EAST in a 6 character field

then FULLNAME = GEOR  EAST  3
```
(2 spaces)

Note the vertical bar is used for concatenation.

If we want to get rid of trailing blanks we could use the double vertical bars ||

Using the same fields as above

```
FULLNAME = PLANT || AREA || '3';
```

would result in FULLNAME = GEOREAST3

8. Reports from Several Files

Data from two or more FOCUS may be joined together by their common values and the result stored in a temporary file. Reports spanning the entire collection of data can then be requested from this 'results' file. In this way we can prepare reports which display information from many files. For example, if we have a SALES file, containing information on product sales, and a COST file, containing information on product costs, we could produce a report on both product sales and costs, instead of one containing only sales or cost information. The joining process is controlled by the FOCUS command JOIN.

The syntax is

JOIN field1 in file1 to field2 in file2 AS joiname

where

'field1' is any field in the file named 'file1'
'field2' is any field in the file named 'file2' (this field must be indexed)

'field1' AND 'field2' MUST CONTAIN THE SAME 'TYPE' OF DATA. For example, the two fields could contain names, some of which might be the same. **IF THE TWO FIELDS CONTAIN THE SAME VALUE (eg. SMITH) THEN THE TWO FILES WOULD BE JOINED FOR THAT RECORD. IF THE FIELD VALUES ARE NOT THE SAME THEN THE TWO FILES ARE NOT JOINED FOR THAT RECORD.** The JOIN command should be issued before entering a report request that accesses data from the joined files.

TABLE and DEFINE commands using the JOINed fields can only be issued after the JOIN command. Fields DEFINEd before a JOIN command are automatically deactivated by issuing the JOIN command. TABLE and DEFINE commands can be applied as if 'file1' were a new file made up of both the original files.

Shown below are two files, the PRODUCT file, and the SALESCOM file, which will be used in the next example.

PRODUCT FILE

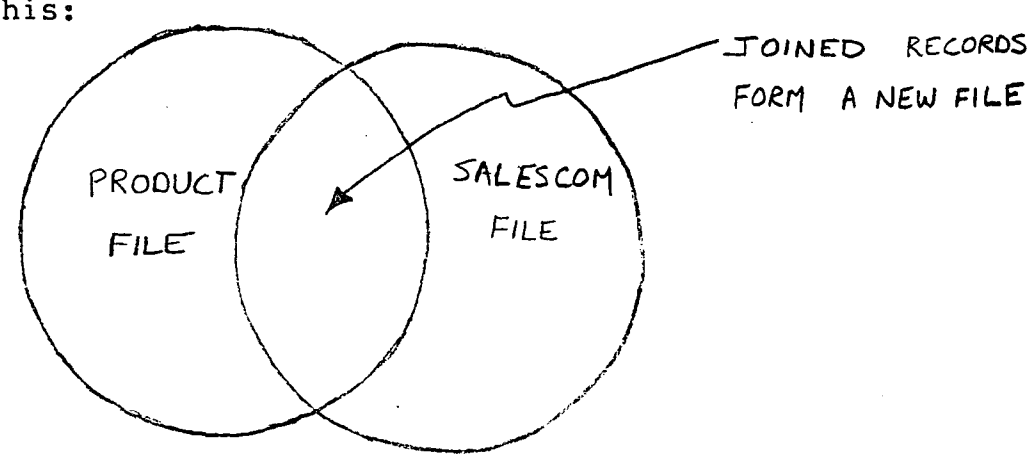
AREA	UNITS	AMOUNT
EAST	1642	7466
WEST	2354	2635
NORTH	5636	1234

SALESCOM FILE

TERRITORY	SALESREP	POINT
EAST	BROWNE	862
EAST	MEHTA	1984
EAST	RUBIN	1482
EAST	VRONSKY	1640
NORTH	SMITH	876

JOIN AREA IN PRODUCT TO TERRITORY IN SALESCOM AS NEW

The above command has the effect of joining the PRODUCT file to the SALESCOM file, which contains information on sales commissions by sales territory. Diagramatically, the JOIN command looks like this:



The records joined are the ones sharing common values in the AREA and TERRITORY fields. For example, all the records containing the value 'EAST' in the PRODUCT file are joined to all the records containing 'EAST' in the TERRITORY file, producing one large file. The same is true for records sharing the value 'NORTH'. As can be seen, not all the records are joined. Notice the WEST area in the PRODUCT file has no match in the SALESCOM file and is therefore not joined.

AREA	UNITS	AMOUNT	SALESREP	POINTS
EAST	1642	7466	BROWNE	862
EAST	1642	7466	MEHTA	1984
EAST	1642	7466	RUBIN	1482
EAST	1642	7466	VRONSKY	1640
NORTH	5636	1234	SMITH	876

After a join, the joined file can be used for reporting

```
JOIN AREA IN PRODUCT TO TERRITORY IN SALESCOM AS NEW
TABLE FILE PRODUCT  <-- (note the use of "file1")
PRINT SALESREP POINTS
BY AREA BY UNITS
END
```

```
AREA    UNITS    SALESREP      POINTS
-----------------------------------

EAST    1642     BROWNE        862
                 MEHTA         1984
                 RUBIN         1482
                 VRONSKY       1640

NORTH   5636     SMITH         876
```

A dynamic JOIN is very useful because it does not affect the master description. New files can be created by joining other files, and these other files are not affected.

APPENDIX SEVEN

PRACTICE PROBLEMS

PRACTICE PROBLEMS

In order to practice the commands you have just learned, you will be asked to write up several small programs. You should then enter these programs into the computer using the TED editor in FOCUS. Run the programs in FOCUS until they work correctly. Solutions are provided following the questions, but please attempt the problem at least once before turning to the solution.

All the questions will be based on a database from a fictitious milk company. The company and the files which make up its database are described below, and on the following page. Let the session leader know when you have successfully finished all the programs.

SAMPLE APPLICATION

Application Description

Our sample application concerns the Milkmore Farms Company. This company manufactures a variety of milk and cheese products and sells them to the public through seven outlet stores. All stores are open seven days a week.

The company wishes to record information about products manufactured and sold. To do this, two files are needed:

1. A Product file to contain descriptive information about each product manufactured by the company. This file is called XPROD.

2. A Sales File to contain sales data about products sold by each outlet store each day. This file is called SALES.

The company routinely adds, updates, and deletes information in these files so that up-to-date product description and sales reports can be produced.

MILKMORE FARMS COMPANY XPROD FILE

Any report request using this file will begin with TABLE FILE XPROD. The following are the fields which make up the file:

1. **PROD_CODE** - A unique alphanumeric code which identifies a certain product. E.g. B10, C17. This field is an indexed field. A field can also be referred to by its alias. ALIAS=PCODE.
2. **PROD_NAME** - The name of the product sold, e.g. whole milk, sour cream. ALIAS=ITEM.
3. **PACKAGE** - Describes the amount of product each package contains, e.g. 16 ounces, 1 dozen. ALIAS=SIZE.
4. **UNIT_COST** - The cost of one package of a product, e.g.$.65, $1.15. ALIAS=COST.

The following are some of the data records in the file:

PROD.CODE	PROD-NAME	PACKAGE	UNIT.COST
B10	WHOLE MILK	16 OUNCES	.65
E1	MEDIUM EGGS	1 DOZEN	.59

MILKMORE FARMS COMPANY SALES FILE

Any report request using this file will begin with TABLE FILE SALES. The fields which make up the file are as follows:

1. **STORE_CODE** - A code which uniquely describes a store which sells Milkmore products, e.g. 14B, 77F. ALIAS=SNO.
2. **CITY** - City in which the store is located. ALIAS=CTY.
3. **AREA** - A letter which describes the area in which the store is located, e.g. S, U. ALIAS=LOC.
4. **DATE** - The date on which the products were sold. ALIAS=DTE.
5. **PROD_CODE** - Same as in XPROD file above. This field is an indexed field.
6. **UNIT_SOLD** - The number of units of a certain product sold. ALIAS=SOLD.
7. **RETAIL_PRICE** - The price the product retails for. ALIAS=RP.
8. **DELIVER_AMT** - The number of units of a product delivered. ALIAS=SHIP.
9. **OPENING_AMT** - The number of units of a product in opening inventory. ALIAS=INV.
10. **RETURNS** - The number of units of a product returned by the customer. ALIAS=RTN.
11. **DAMAGED** - The number of units of a product which are damaged. ALIAS=BAD.

The following are some of the data records in the file:

STORE CODE	CITY	AREA	DATE	PROD.CODE	UNIT.SOLD	RETAIL_PRICE	DELIVER_AMT	OPENING_AMT
14B	STAMFORD	S	12/12	B10	60	$.95	80	65
14B	STAMFORD	S	12/12	B12	40	$1.29	20	50
K1	NEWARK	U	10/18	B10	13	$.99	30	15
K1	NEWARK	U	10/19	B12	29	$1.49	30	30

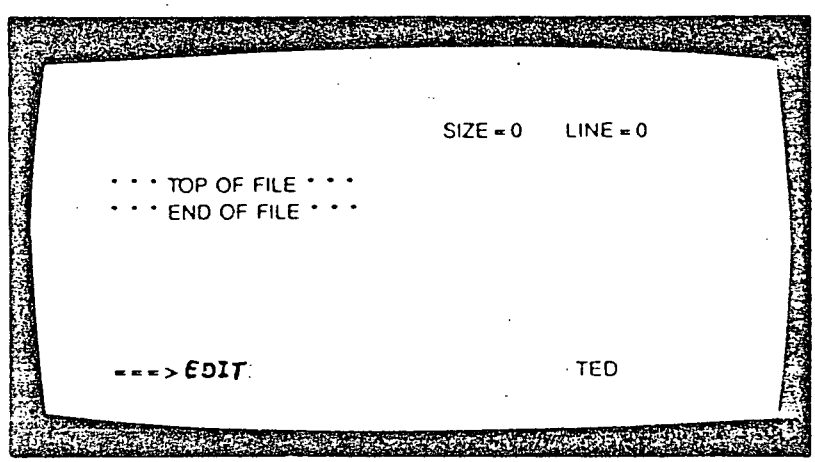
EXAMPLE OF A FOCUS SESSION

If we want to create a program to display a store's identification code, and the number of units of each product sold at the store, and want to call it SHOWSALES, the FOCUS session would appear as follows:

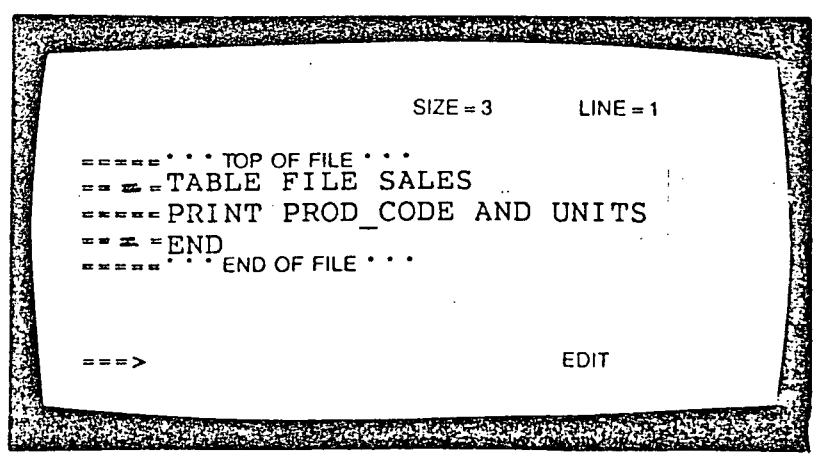
>>TED SHOWSALES.FEX

This command puts us into the TED editor where we can enter our program. The FEX extension indicates that the program is a FOCUS execution procedure. It must be added to all program names.

Once in TED the following screen appears. To create or edit our program we enter **EDIT** at the command line (bottom of the screen)



Now we want to create space between the TOP OF FILE and END OF FILE in order to be able to enter our program. To do this we issue the command ADD 3 at the command line. We now enter the program in the space just provided using the arrow keys, and the double arrow key as the return key.



Once we are finished entering the program we return to the command line using the return key. There we type the command FILE, to save the program we have just entered. This command also transfers us out of the TED editor back into normal FOCUS mode. To run the program, we can now type EX and the name of the program.

```
>>EX SHOWSALES.FEX
```

If an error occurs in the program, a message will appear. At this point you will usually be prompted by a single caret ">". If you do not wish to continue execution of the incorrect program, the word **QUIT** can be typed as a response and the report request is cancelled. If there is more than one error, you may have to type QUIT to two ">" prompts in order to get back to normal FOCUS mode ">>". Once back in FOCUS you can edit your program with the TED editor (TED SHOWSUM.FEX).

If the program works, FOCUS will tell us the length of the report and ask us to hit the return key to see the report. Hitting the return key shows us the first page of the report. If the report occupies more than one page we would keep hitting the return key to see the rest of the report. At the end of the report, an END-OF-REPORT message will appear at the bottom of the screen. Hitting the return key two or three more times will get us back to the FOCUS prompt '>>'.

PRACTICE PROGRAMS FOR THE REPORT GENERATION SESSION

1. Produce a report which shows the total units sold and total returns for each store_code. The first part of the output should appear as follows:

STORE_CODE	UNIT_SOLD	RETURNS
14B	347	38

2. Produce report which will sum the units sold for each date within each city, and prints these sums in alphabetical order by city. The first part of the report should appear as follows:

CITY	DATE	UNIT_SOLD
NEW YORK	10/17	162

3. Produce a report which will sum the units sold for each city and prints these horizontally across the top of the page. The column headings should appear as:

CITY NEW YORK	NEWARK	STAMFORD	UNIONDALE

4. Produce the following matrix-type report which details units sold, and returns for each city by retail price. (Note- there may not be units sold, in each city, at every retail price. In this case a dot will occur in place of a number)

	CITY NEW YORK		NEWARK	
RETAIL_PRICE	UNIT_SOLD	RETURNS	UNIT_SOLD	RETURNS
.85	30	2	.	.

5. Produce a report which shows the quantities of a specific product (b10) sold, and on hand (opening amt) in a specific city (Newark). The first part of the report should appear as:

CITY	PROD-CODE	UNIT-SOLD	OPENING-AMT
NEWARK	B10	30	10

6. Produce a report which shows the units sold for each product code by city. For each city, show the units sold by date. Show subtotals for both the city, and each date within the city. Also show a grand total for units sold. The report should have the following structure:

CITY	DATE	PROD-CODE	UNIT-SOLD
NEW YORK	10/17	B10	30
		:	:
		:	:
		:	:
* TOTAL DATE 10/17			162
* TOTAL CITY NEW YORK			162

7. Produce a report which shows the ratio of returns to units sold for each product code. The first part of the report should appear as follows:

PROD-CODE	RATIO
B10	.17

8. Produce a report which prints the product name, unit cost, retail price, and ratio of cost to retail price for each product. The first part of the report should appear as follows:

PROD-NAME	UNIT-COST	RETAIL-PRICE	RATIO
WHOLE MILK	$.65	$.95	.68

SOLUTION TO PROBLEM 1

```
TABLE FILE SALES
SUM UNIT_SOLD RETURNS
BY STORE_CODE
END
```

```
PAGE      1

STORE_CODE   UNIT_SOLD   RETURNS
----------   ---------   -------
14B              376         40
14Z              162         15
77F               65          1
K1                42          2
```

SOLUTION TO PROBLEM 2

```
TABLE FILE SALES
SUM UNIT_SOLD
BY CITY BY DATE
END
```

PAGE 1

CITY	DATE	UNIT_SOLD
NEW YORK	10/17	162
NEWARK	10/18	13
	10/19	29
STAMFORD	12/12	376
UNIONDALE	10/18	65

SOLUTION TO PROBLEM 3

```
TABLE FILE SALES
SUM UNITS ACROSS CITY
END
```

PAGE 1

CITY NEW YORK	NEWARK	STAMFORD	UNIONDALE
162	42	376	65

SOLUTION TO PROBLEM 4

```
TABLE FILE SALES
SUM UNIT_SOLD RETURNS
ACROSS CITY
BY RETAIL_PRICE
END
```

PAGE 1.1

	CITY					
	NEW YORK		NEWARK		STAMFORD	
RETAIL_PRICE	UNIT_SOLD	RETURNS	UNIT_SOLD	RETURNS	UNIT_SOLD	RETURNS
$.85	30	2	.	.	.	.
$.89	30	4	.	.	.	.
$.95	.	.	.	.	60	10
$.99	.	.	13	1	80	9
$1.09	35	4	.	.	70	8
$1.29	.	.	.	.	40	3
$1.49	.	.	29	1	.	.
$1.89	20	2	.	.	29	2
$1.99	15	0	.	.	25	3
$2.09	32	3	.	.	.	.
$2.19	.	.	.	.	27	0
$2.39	.	.	.	.	45	5
$2.49	.	.	.	.	.	.

SOLUTION TO PROBLEM 5

```
TABLE FILE SALES
PRINT CITY PROD_CODE UNIT_SOLD OPENING_AMT
IF PROD_CODE EQ B10
IF CITY EQ  NEWARK
END
```

PAGE 1

CITY	PROD_CODE	UNIT_SOLD	OPENING_AMT
NEWARK	B10	13	15

SOLUTION TO PROBLEM 6

```
TABLE FILE SALES
PRINT PROD_CODE UNIT_SOLD
BY CITY BY DATE
ON DATE SUB-TOTAL
END
```

PAGE 1

CITY	DATE	PROD_CODE	UNIT_SOLD
NEW YORK	10/17	B10	30
		B17	20
		B20	15
		C17	12
		D12	20
		E1	30
		E3	35
*TOTAL DATE 1017			162
*TOTAL CITY NEW YORK			162
NEWARK	10/18	B10	13
*TOTAL DATE 1018			13
	10/19	B12	29
*TOTAL DATE 1019			29
*TOTAL CITY NEWARK			42
STAMFORD	12/12	B10	60
		B12	40
		C13	25
		C7	45
		D12	27
		E2	80
		E3	70
*TOTAL DATE 1212			376
*TOTAL CITY STAMFORD			376
UNIONDALE	10/18	B20	25
		C7	40
*TOTAL DATE 1018			65
*TOTAL CITY UNIONDALE			65
TOTAL			616

SOLUTION TO PROBLEM 7

```
DEFINE FILE SALES
RATIO=RETURNS/UNIT_SOLD;
END
TABLE FILE SALES
PRINT PROD_CODE RATIO
END
```

PAGE 1

PROD_CODE	RATIO
B10	.17
B12	.07
C13	.12
C7	.11
D12	.00
E2	.11
E3	.11
B10	.07
B17	.10
B20	.00
C17	.00
D12	.15
E1	.13
E3	.11
B20	.04
C7	.00
B12	.03
B10	.08

Solution to Problem 8

```
JOIN PROD_CODE IN XPROD TO PROD_CODE IN SALES AS NEW
DEFINE FILE XPROD
RATIO = UNIT_COST / RETAIL_PRICE;
END
TABLE FILE XPROD
PRINT PROD_NAME UNIT_COST RETAIL_PRICE RATIO
END
```

APPENDIX EIGHT

QUESTIONNAIRE

FOURTH GENERATION LANGUAGE EXPERIMENT

Please fill out all sections of the questionnaire as accurately as possible. The information you provide will be used to determine factors which could explain your success in the experiment. If you feel their are other factors which could be important in explaining your success with a fourth generation language, and are not mentioned below, please include them at the end of the questionnaire. All information will be held strictly confidential. If you have any questions do not hesitate to ask the lab assistant. Thank you for your cooperation.

1. Name _____

Please complete question 2 if you are now a student.
In addition, please complete question 3 if you have worked full time, or now work full time.

2. Which school do you attend? (please check one)

UBC.....
BCIT.....
Other (please specify)_____

3. a) Which company did/do you work for? _____

b) What was your job title? _____

c)In your work, did/do you make use of computers? YES ____ NO _____

d) If yes, what type of work do/did you do with the computer? (check any of the following which apply)

Querying databases.....
Data entry.....
Programming.....
Software user.....
Computer operator.....
Other, please specify _____

e) If your job involves/involved programming, for how many years have/had you been programming at work? (indicate number of years)

..... year(s)

f) If your job involves/involved programming, approximately what percentage of your time at work is/was spent using computers for this purpose? (indicate percentage)

..... percent

4. What is your program of study at school? (check one of the following)

Commerce.....
M.B.A. .....
Computer Science.....
Other (please specify)_____

5. If you have previous degrees/diplomas, please specify the title of the degree

_____

_____

_____

6. Have you ever done any computer programming? YES ..... NO .....

If yes, list the programming languages you know, the number of years experience you have had with each, and the approximate number of programs you have written in each.(eg. COBOL, FORTRAN, APL, PASCAL)

Language	Years Experience	Approx. no. of programs written
i)		
ii)		
iii)		
iv)		
v)		
vi)		
vii)		
viii)		
ix)		

---

7. Have you ever used any report writers, spreadsheets, query languages, database management systems, or fourth generation languages? (eg. Sequel, EDBS, Dbasell, Lotus 1-2-3, IMS, FOCUS, RAMIS, TOTAL, DB2, IDEAL, ADF, ADABAS)

YES.....
NO.....

If yes, please list them below.

Software	Years Experience	Approx. no. of programs written
a.		
b.		
c.		
d.		
e.		
f.		
g.		
h.		
i.		
j.		

---

8. How would you characterize your use of microcomputers over the past few years ? (please circle the number which best describes your use)

1 2 3 4 5 6 7
Never use Use every day

9. List below any other factors in your background which you think might have had an effect on your performance in the experiment.

_____

_____

_____

_____

_____

Thats it, thank you for your participation. Please do not discuss this study with other participants as you may unduly influence their performance and learning process.

APPENDIX NINE

ESTIMATION OF SAMPLE SIZE NEEDED

APPENDIX 9 - ESTIMATION OF THE SAMPLE SIZE NEEDED

The two methods of estimating the sample size needed are, the power approach and the estimation approach. The power approach uses an estimate of the standard deviation (σ), the level at which Type I (α) and Type II (β) errors are to be controlled, and an estimate of the magnitude of the minimum range (Δ) of the factor level means (μ) which is important to detect with high probability. The estimation approach specifies the major comparisons of interest, and from these, determines the expected widths of the confidence intervals for various sample sizes, given an advance planning value for the standard deviation. The approach is iterative, starting with an initial guess for the needed sample sizes. If the confidence intervals, based on the initial sample sizes are satisfactory, the iteration process is terminated.

The power method was used first to determine a range of likely sample sizes. A sample size was then used in the estimation approach to ensure that the confidence intervals were satisfactory. The calculations are shown below.

POWER APPROACH

An estimate of the standard deviation of the subject population is needed for determining the sample size in this approach. The mean and standard deviation from the pilot study were 75.4 and 13 respectively. This standard deviation will be used as our estimate. We would like our hypotheses tests to detect differences in mean scores between subjects of about 10 marks (Δ=10). Any difference smaller than 10 could be due to

chance. Because σ=13, and we need an even Δ/σ ratio to use the statistical tables, we will increase Δ slightly to 13. The number of levels of the first factor (complexity) is a=2, and the number of levels of the second factor (experience) is b=2. We will control the Type I and Type II errors at α=.05 1 - β=.90

The power method says that if we let the number of factor levels (r) equal the number of factor levels of the first factor (a) then the resulting sample size equals the number of levels of the second factor (b) multiplied by the sample size for each treatment (n). From the power tables [From Table A-10 in Neter and Wasserman, Applied Linear Statistical Models] this resulted in an n of 11.5. By increasing the difference to Δ=16.25, the sample size becomes 7.5. On the other hand, with 1 - β=.95 a sample size of 9 is needed.

From the above calculations, it appears we need a sample size of approximately 8 to 12 for each treatment. Since there are four treatments, this results in a total sample size of 32 to 48.

ESTIMATION APPROACH

We can now use our estimates from the power approach to check the confidence intervals obtained for hypotheses testing.

HYPOTHESIS ONE

Empirically, the first hypothesis, that experienced programmers will obtain higher mean scores than novices on simple and complex tests, involves a contrast of factor level means.

We must determine if the confidence interval for a given sample size is sufficiently small for our analysis. The confidence interval for contrast of factor level means is L +/- t [1 - α; (n - 1) x (a x b)] s(L) where L is the difference between factor level means. L is the estimator of L. L is the difference in factor level sample means. s (L) is the standard deviation of L and can be computed as the square root of $2\sigma/n$.

Therefore the confidence interval, if n=12, will be L +/- 2.47. This is a sufficiently small interv al for estimating the difference in scores between experienced programmers and novices, as we were willing to accept a Δ=13.

HYPOTHESIS TWO

The second hypothesis, that the difference in scores between complex and simple tests will be greater for novices than experienced programmers, involves a contrast of treatment means. If n=12, the confidence interval for a contrast of treatment means with interaction is L +/- t[1-α; (n-1)ab]s(L), where L = $[(Y_{11} - Y_{12}) - (Y_{21} - Y_{22})]$ and $s^2(L) = MSE/n\Sigma\Sigma c_{ij}^2$ = 4.33, s(L) = 2.08. With n=12, and α = .05, L +- 3.49.

Therefore the confidence interval is +/- 3.49 and is sufficient. Therefore n=12 is a sufficient sample size, which makes our total target sample N=48.

APPENDIX TEN

DATA COLLECTED DURING THE EXPERIMENT

OBS	SN	EI	PWE	YPEW	MBA	CSS	OS	E3GL	N3GL	N3GLW	RWPW	QLPW	N4GLW	TTIME	SIMP	RT	PT	TT	Q1	Q2	Q3	OV	EXP
1	1	0	0	0.00	0	1	0	1	6	25	0	0	0	204	0	44	115	45	43	81	33	47	0
2	2	0	0	0.00	0	1	0	1	7	64	0	0	1	138	0	30	78	30	73	62	41	47	1
3	3	1	0	0.00	1	0	0	1	1	6	0	0	0	235	0	80	110	45	100	73	19	34	0
4	4	0	0	0.00	0	1	0	1	6	54	0	0	25	133	0	25	69	39	90	100	66	76	1
5	5	1	0	0.00	1	0	0	1	1	3	4	0	0	176	1	40	91	45	63	52	13	40	0
6	6	1	0	0.00	1	0	0	1	2	55	5	0	0	151	0	46	60	45	83	92	66	73	1
7	7	0	0	0.00	0	1	0	1	6	78	2	1	0	131	0	22	64	45	100	88	23	42	1
8	8	0	1	0.20	0	1	0	1	6	30	0	0	2	134	0	33	63	38	97	100	73	81	.
9	9	0	0	0.00	0	1	0	1	5	41	0	0	3	143	1	26	75	42	90	100	68	84	.
10	10	0	0	0.00	0	1	0	1	4	35	0	1	3	146	1	31	89	26	100	96	79	90	.
11	11	1	0	0.00	1	0	0	1	2	9	8	0	0	156	1	52	59	45	100	100	61	84	0
12	12	0	0	0.00	0	1	0	1	5	39	0	0	3	138	0	31	65	42	95	27	72	59	.
13	13	1	0	0.00	1	0	0	1	2	5	0	0	0	150	1	45	60	45	93	60	18	54	0
14	14	0	0	0.00	0	1	0	1	6	30	0	0	2	150	0	31	82	37	57	88	47	59	.
15	15	1	1	0.15	0	0	1	1	3	33	0	0	0	190	1	70	80	40	100	100	76	90	1
16	16	0	0	0.00	0	1	0	1	6	35	0	0	3	155	0	32	79	44	97	38	84	71	.
17	17	0	0	0.00	0	1	0	1	6	47	0	0	0	152	0	25	82	45	97	77	25	40	1
18	18	1	0	0.00	1	0	0	1	4	20	4	0	0	188	0	53	90	45	97	58	52	53	0
19	19	0	1	0.13	0	1	0	1	6	51	0	0	40	136	1	23	78	35	100	100	79	91	1
20	20	1	0	0.00	0	0	1	1	3	10	1	0	0	155	1	37	75	43	100	100	79	91	0
21	21	0	0	0.00	0	1	0	1	5	28	0	1	1	157	0	32	89	36	92	38	28	31	.
22	22	1	0	0.00	1	0	0	1	2	13	2	0	0	169	1	66	73	30	50	92	74	71	0
23	23	1	0	0.00	1	0	0	0	0	0	5	0	0	154	1	39	70	45	90	100	74	86	0
24	24	0	1	0.25	0	1	0	1	5	74	0	0	1	159	0	22	96	41	80	42	58	53	1
25	25	1	0	0.00	1	0	0	1	1	7	7	0	0	183	1	57	81	45	100	16	76	68	0
26	26	0	0	0.00	0	1	0	1	5	10	0	0	0	158	1	40	82	36	67	68	74	70	0
27	27	0	0	0.00	0	1	0	1	7	39	0	0	2	163	0	28	96	39	100	58	41	46	1
28	28	1	1	1.00	1	0	0	0	5	23	0	0	0	130	1	40	60	30	93	80	53	73	1
29	29	1	0	0.00	1	0	0	1	3	23	12	0	0	158	0	43	70	45	100	4	63	46	0
30	30	1	0	0.00	0	0	1	0	0	0	0	0	0	182	0	52	85	45	93	27	61	53	0
31	31	1	0	0.00	1	0	0	1	2	33	3	1	0	190	1	75	80	35	100	100	100	100	0
32	32	0	0	0.00	0	1	0	1	6	27	0	0	1	166	0	22	105	39	100	38	44	42	.
33	33	0	0	0.00	0	1	0	1	5	38	0	0	6	169	1	33	104	32	87	60	32	57	.
34	34	1	1	0.40	0	0	1	1	3	32	0	0	0	200	0	60	95	45	70	77	28	42	1
35	35	0	0	0.00	0	1	0	1	5	87	0	0	25	173	1	33	97	43	83	.	61	71	1
36	36	1	0	0.00	1	0	0	0	0	0	0	0	0	199	1	50	104	45	63	0	0	20	0
37	37	0	0	0.00	0	1	0	1	7	21	1	0	2	174	1	52	94	28	100	80	50	74	0
38	38	1	0	0.00	1	0	0	0	0	0	0	0	0	188	1	45	103	40	100	28	92	77	0
39	39	0	1	0.30	0	1	0	1	4	13	0	0	20	180	0	38	97	45	97	4	39	29	.
40	40	1	0	0.00	1	0	0	0	0	0	0	0	0	198	1	60	93	45	83	76	50	68	0
41	41	1	0	0.00	1	0	0	1	2	5	7	0	0	162	1	53	64	45	100	52	50	67	0
42	42	0	0	0.00	0	1	0	1	5	37	1	0	2	185	1	42	101	42	100	88	84	90	.
43	43	1	0	0.00	0	0	1	0	0	0	0	0	0	152	0	45	62	45	100	73	47	54	0
44	44	1	0	0.00	0	0	1	1	5	46	3	0	0	125	1	45	50	30	100	100	76	90	.
45	45	0	0	0.00	0	1	0	1	6	143	11	0	0	184	0	44	95	45	30	58	53	54	1
46	46	0	0	0.00	0	1	0	1	5	23	1	0	1	175	0	50	80	45	100	96	70	78	0
47	47	1	0	0.00	1	0	0	0	0	0	0	0	0	145	0	60	45	40	93	77	77	77	0
48	48	1	0	0.00	1	0	0	1	2	7	16	0	5	190	0	60	48	48	93	0	33	23	0
49	49	0	0	0.00	0	1	0	1	5	15	0	0	3	175	0	60	70	45	83	85	84	84	0
50	51	0	0	0.00	0	1	0	1	5	11	1	0	0	165	0	40	90	35	87	58	48	51	0
51	53	0	0	0.00	0	1	0	1	5	27	0	0	0	145	1	40	75	30	90	100	55	78	.
52	55	0	0	0.00	0	1	0	1	5	24	0	0	0	154	0	44	65	45	100	81	84	83	0
53	56	0	0	0.00	0	1	0	1	7	320	0	1	1	149	1	44	75	30	100	96	71	87	1
54	57	0	0	0.00	0	1	0	1	6	40	0	0	2	174	0	44	90	40	100	88	52	62	.
55	58	0	0	0.00	0	1	0	1	4	19	0	2	1	179	1	44	90	45	100	100	97	99	0
56	59	0	0	0.00	0	1	0	1	7	69	0	0	0	194	1	44	105	45	90	92	63	80	1
57	60	0	0	0	0	1	0	1	6	29	0	0	0	199	1	49	105	45	82	40	0	37	0