

Learning programming in computer laboratories : a case study
by Reggie Ching-Ping Kwan

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Education in Adult and Higher Education
Montana State University
© Copyright by Reggie Ching-Ping Kwan (1997)

Abstract:
The Computer Science Department at Montana Tech of the University of Montana has designed and implemented a programming laboratory for the introduction to computer science course. The purpose of this study was to investigate how students utilized the newly designed laboratory in learning how to program and to analyze the strengths and weaknesses of the setup in the laboratory physically as well as different teaching and learning activities in the lab.

The study was done by observing approximately 150 students in the laboratory for 15 weeks. Assessment surveys were administered in the beginning of the semester and again at the end. Two rounds of in-depth interviews were conducted in the middle of the semester and then again at the end with 21 participants.

The study concentrated on students' learning strategies and lab learning activities. Results from the survey and interviews indicated the laboratory portion of the course was a major part of students' learning.

The study also revealed the importance of the physical layout of the laboratory. Most students preferred working alone or having a partner with similar prior experience. Students also considered classmates in their vicinity to be a good source for discussions. Most students felt comfortable seeking help from lab assistants. Gender made a small difference in terms of the number and the type of questions asked. All students considered printed lab manuals to be useless and preferred on-line manuals. The language C and the Turbo programming environment did not present any problem in the lab. The Turbo debugger was the most popular tool in the Turbo environment.

The lab activities record and explain as well as experiment and discover were well received. The design and justify activities received some complaints and caused most problems in students' lab reports. However, students regard all three activities instrumental in their learning. The time needed for the design and justify activity was unpredictable.

Many participants of this study suggested a different format of the lab. Some also recommended modifications of the lab report especially the write-up portion.

LEARNING PROGRAMMING IN COMPUTER LABORATORIES

-- A CASE STUDY

by

Reggie Ching-Ping Kwan

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Education

in

Adult and Higher Education

MONTANA STATE UNIVERSITY
Bozeman, Montana

December 1997

D378
K979

APPROVAL

of a thesis submitted by

Reggie Ching-Ping Kwan

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

12-19-97
Date

Gary J. Conti
Chairperson, Graduate Committee

Approved for the Major Department

January 5, 1998
Date

Lorna A. Rugg
Head, Major Department

Approved for the College of Graduate Studies

1/15/98
Date

Joseph J. Fedock
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a doctoral degree at Montana State University--Bozeman, I agree that the Library shall make it available to borrowers under rules of the Library. I further agree that copying of this thesis is allowable only for scholarly purposes, consistent with the "fair use" as prescribed in the U.S. Copyright Law. Requests for extensive copying or reproduction of this thesis should be referred to University Microfilms International, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom I have granted "the exclusive right to reproduce and distribute my dissertation in and from microfilm along with the non-exclusive right to reproduce and distribute by abstract in any format in whole or in part."

Signature Reggie Kwan

Date 1/9/98

TABLE OF CONTENTS

	page
LIST OF TABLES	viii
LIST OF FIGURES	ix
ABSTRACT	x
1. INTRODUCTION	1
Living with Computers	1
Computer Programming	4
An Introductory Course in Computer Science	7
Montana Tech's Computer Science Program	11
The New CS1 course at Montana Tech	12
The New Laboratory in CS1	14
Problem	16
Purpose	17
Research Questions	17
Significance of Study	17
Definition of Terms	19
Limitations	21
Delimitations	21
Assumptions	21
2. BACKGROUND AND REVIEW OF LITERATURE	23
History of Computing	23
The Birth of the Computer	23
Generations of Languages and Hardware	25
The C Language	30
Computing As A Discipline	33
Laboratory Activities	35
Record and Explain	35
Experiment and Discover	38
Design and Justify	40
Computers and Cognition	42
Technology and Education	42
Computers in Education	44

TABLE OF CONTENTS--Continued

page

Adult Learners in Computing Science..... 45
Educational Objectives..... 47
Model of Instructions: Andragogy and Pedagogy..... 50
Laboratory Learning..... 53

3. METHODOLOGY..... 55

Naturalistic Inquiry..... 55
Case Studies..... 59
Research Population..... 61
Procedures..... 64
Observation..... 66
The Physical Environment..... 66
The Programming Environment..... 69
Teams..... 72
Lab Assistants..... 72
Interview Questions..... 73

4. FINDINGS..... 78

Beginning of Semester Survey..... 78
End-of-Semester Survey..... 79
Observation and Interview Results 80
The Physical Environment..... 81
The Circular Lab..... 81
The Rectangular Lab..... 84
The Programming Environment..... 87
Learning Strategies..... 89
Working in Teams..... 89
Less Experienced with More Experience Team Pairings..... 91
Similar Experience Team Pairing..... 94
Lab Assistants..... 96
Students' Point of View..... 97
Lab assistants' Point of View..... 99
Lab Manuals..... 101
The Time Factor..... 102
Background in Mathematics..... 108

TABLE OF CONTENTS--Continued

page

Cooperative Learning Environment..... 111
Gender Differences..... 113
The Age Factor..... 115
The Language C..... 117
Write-ups..... 119
Teaching and Learning Activities..... 123
Record and Explain..... 123
Experiment and Discover..... 126
Design and Justify..... 128
Interview Summary..... 132

5. SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS..... 134

Summary..... 134
Conclusions and Recommendations..... 137
Beginning-of-Semester Assessment..... 138
End-of-Semester Assessment..... 139
Partners and Neighbors..... 139
Physical Environment..... 141
On-line Help Versus Printed Manual..... 141
Post-lab Write-ups..... 142
Teaching and Learning Activities..... 143
Lab Assistants..... 144
The Role of Lab Assistants..... 144
Weekly Meetings..... 144
Same Day Labs..... 145
Recommendations for Further Research..... 146
The Future..... 147

REFERENCES CITED..... 149

APPENDICES..... 159
Appendix A Beginning-of-Semester Survey..... 160
Appendix B End-of-Semester Survey..... 162
Appendix C Lab Exercise with Record and Explain Activities..... 164
Appendix D Lab Exercise with Experiment and Discover Activities..... 167

TABLE OF CONTENTS--Continued

page

Appendix E Lab Exercise with Design and Justify Activities..... 169

LIST OF TABLES

Table		Page
1.	Generations of Computer Hardware.....	29
2.	Assumptions of the Andragogical and Pedagogical Models.....	51
3.	Program Design for the Pedagogical Model and Andragogical Model.....	52
4.	Profiles of Students.....	62
5.	End-of-Semester Survey.....	80

LIST OF FIGURES

Figure		Page
1.	Machine and Assembly Languages.....	26
2.	The Result of the Program Logical And.....	36
3.	Scenarios of a Ham and Cheese Sandwich.....	37
4.	The Result of the Program Logical Or.....	39
5.	Bloom's Classification of Educational Objectives.....	47
6.	Steinaker's Classifications of the Experimental Domain.....	48
7.	Gagne's Learning Hierarchy.....	50
8.	The Layout of the Rectangular Lab.....	68
9.	The Layout of the Circular Lab.....	70
10.	Program Segment that Produced the Wrong Sum.....	124
11.	Program Segment that Produced the Right Sum.....	126
12.	Algorithm 1 for the Lottery Program.....	129
13.	Algorithm 2 for the Lottery Program.....	130

ABSTRACT

The Computer Science Department at Montana Tech of the University of Montana has designed and implemented a programming laboratory for the introduction to computer science course. The purpose of this study was to investigate how students utilized the newly designed laboratory in learning how to program and to analyze the strengths and weaknesses of the setup in the laboratory physically as well as different teaching and learning activities in the lab.

The study was done by observing approximately 150 students in the laboratory for 15 weeks. Assessment surveys were administered in the beginning of the semester and again at the end. Two rounds of in-depth interviews were conducted in the middle of the semester and then again at the end with 21 participants.

The study concentrated on students' learning strategies and lab learning activities. Results from the survey and interviews indicated the laboratory portion of the course was a major part of students' learning.

The study also revealed the importance of the physical layout of the laboratory. Most students preferred working alone or having a partner with similar prior experience. Students also considered classmates in their vicinity to be a good source for discussions. Most students felt comfortable seeking help from lab assistants. Gender made a small difference in terms of the number and the type of questions asked. All students considered printed lab manuals to be useless and preferred on-line manuals. The language C and the Turbo programming environment did not present any problem in the lab. The Turbo debugger was the most popular tool in the Turbo environment.

The lab activities record and explain as well as experiment and discover were well received. The design and justify activities received some complaints and caused most problems in students' lab reports. However, students regard all three activities instrumental in their learning. The time needed for the design and justify activity was unpredictable.

Many participants of this study suggested a different format of the lab. Some also recommended modifications of the lab report especially the write-up portion.

CHAPTER 1

INTRODUCTION

Living with Computers

Television, telephone, automobiles, and other modern technological developments have revolutionized the way people live, work, and play. The computer is doing no less.

The typical computer user today is no longer the stereotype Ph.D. who works in an underground laboratory with a 4-foot steel door. Whether making a phone call, driving an automobile, or adjusting a programmable thermostat, people unknowingly use the computer and its programs. The benefits of using computers are taken for granted. Unless something goes wrong, people seldom realize how much they rely on computers. In fact, it is hard to escape computers, and they are even changing the way people think.

Computers provide tools that most people cannot imagine to live without. These include devices such as word processors, database systems, multi-media systems, electronic spreadsheets, desktop publishing, and the Internet. These are tools that professional use.

The impacts of computers on sociological aspects of the human condition has drawn much attention. Jastrow tried to address the growth of computing power and relate this growth to human evolution (1987, pp. 512-513) by pointing out that computers are part of many jobs, if not all. Computers monitor our financial activities, provide diagnoses and treatments, guide missiles, print payroll checks; the list goes on and on. Information is being exchanged electronically around the world at an astonishing speed. People have the opportunity to be more "informed" than at any time in history. The future may, indeed, consist of symbiotic relationship where computers minister to human's social and economic needs.

Besides speed and storage, computers are acquiring more and more human capabilities such as speech and virtual reality, or other simulation-related advances. The technology of computing is moving so fast that the present way of doing computing such as the using a keyboard could be rapidly becoming obsolete (Kahn, 1996, p. 49). As user-computer communication becomes more natural, some people find, especially the "MTV generation," computers so interesting and stimulating that they prefer the company of computers over human (Saffo, 1994, pp. 16-17).

Living with the good provided by computers also means living with the dangers created by them. Like any major innovation, computers are not without problems. Computer game addiction, electronic embezzlements, as well as relying too much on computers are some of the hazards of living with them.

The trust society places in computers should be alarming. It is one thing to rely on a computer to add up scores for the Miss America Pageant, but trusting a computer to decide cases in "the computer court" is another (Gersting & Gemgnani, 1988, pp. 270-271).

With both this good and the bad characteristics, computers are everywhere, and their use is increasing. Regardless of the potential harms of computers, they can no doubt help in every imaginable way, and are stimulating diverse changes in society. For example, to compete with MTV on a fair playing field, teachers from K-12 have started using digitized motion video in the classroom (Mageau, 1990, p. 27). The Internet has already become an indispensable learning tool because it is such a massive information delivery vehicle (Parker, 1996, p. TSW 1-11). It is obvious that they are unavoidable and play an

important role in daily life. Consequently, people ought to know more about them.

Computer Programming

For a computer to function, it must have its hardware and software working properly. While hardware refers to physical devices, software consists of the non-physical parts or programs. Programs are step by step instructions that direct the computer to do the tasks desired by the operator and to produce the desired results.

Programming, the writing of software, is a process which programs are designed, written, and tested. It was once considered an art, sometimes a dark art, which was understood only by a brilliant few who took great pride in their craft and which others could not comprehend. However, as one of the most famous computer scientists wrote as the first statement of his seven-volume series, entitled The Art of Computer Programming,

> The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music. (Knuth, 1973, p. v)

Though his books became the "bibles" of computer science the art of programming has migrated, through the ongoing revolution of computing, into a mixture of art, craft, and science in the 1980's (Starkey & Ross, 1984, p. 1). The revolution continues and so does the debate of what computer science is. As far back as 1971, Weinberg pointed out that software should be developed according to the ego-less programming paradigm (pp. 47-65). It was suggested that programs should be developed according to standardized methods that are understandable to keep "artist" from developing "in-maintainable" programs. Weinberg even went as far as describing programming as a "social activity" (1971, pp. 67-93). Nevertheless, even in the design stages today, there are numerous methods, both simple and sophisticated, to describe a solution. Methods range from the traditional flow-charts to the formal and mathematical description of Z-method (Abrial, 1980), and Class/Responsibilities/Collaborators (CRC) Cards (Booch, 1994, p. 159, pp. 237-239, Lorenz, 1993, p. 118) in object oriented programming. Compared to the early days, programming takes a very different approach today (Dijkstra, 1980, pp. 571-572). New methods still come up constantly. As the 1989 report of the Association for Computing

Machinery (ACM) Education Board asked, "Is computer science a science? An engineering discipline?" (p. 9). Even the Institute of Electrical and Electronic Engineers (IEEE) is having more and more influence on computer science curriculum. The close association between these two is demonstrated by the fact that more than 35 universities with computer science programs recently added a software engineering program in the United States (Gibbs, 1989, pp. 601-604), and most of them still have strong ties with their respective computer science program.

Programming is a major part of computer science or software engineering. The most common introductory course in the computer science and software engineering curricula is programming (p. 9). It is true that computer science encompasses far more than programming, and yet every computing major should achieve competence in it (p. 11).

Basic elements of learning how to program includes, at least, the learning of a programming language and problem-solving skills in computing. Designing, implementing, testing, and maintaining a program are all activities in programming. Thus, computer programming is a continuous process in which problem or problems are solved through some systematic steps. The steps are usually referred to as the

"software life cycle." The programming language is used solely in the implementation step of the cycle.

An Introductory Course in Computer Science

CS 1, Computer Programming I, introduced by ACM Curriculum Committee (ACM, 1978) is a generic course for introduction to computer science in most colleges. It covers both problem solving in computing and at least one high-level language. A high-level language in computing refers to a programming language which resembles natural languages while a low-level language refers to a programming language which is close to machine codes, i.e. 1's and 0's. Problem solving skills in CS 1 are the development of simple algorithms which are step by step solutions to problems. A good algorithm described in pseudolanguage program (Starkey & Ross, 1984, pp. 41-43) can be easily translated into a high-level language. Students are expected to study existing algorithms as well as develop their own. They then implement the algorithms into a target language.

Although the first electronic computer was introduced over 40 years ago, most computer science departments were founded only in the last 20 years. A computer science department usually emerged from a mathematics department,

and many are still part of a mathematics department. Once in a while, a computer science program is offered through the college of engineering or business.

CS 1 has gone through major changes in both the languages used and the concepts covered in the last 18 years since the first comprehensive guideline from ACM. However, one thing which has remained quite constant is the historic tie between mathematics and computer science. As a result, courses in computer science are taught very much like mathematics. Instructors lecturing and students taking note passively are common scenes in computer science courses.

Despite this historic linkage, a fundamental separation between computer science and mathematics is inevitable at least in ideology because "curriculum needs often stem from the nature of the content itself" (Conti & Fellenz, 1991, p. 21). Furthermore, "teachers who do not vary their strategies according to the content will fail to stimulate the participants sufficiently to achieve their teaching goals" (Seaman & Fellenz, 1988, p. 15). Even some mathematics professors are using packages like Mathematica or Maple to better the learning of different mathematical concepts.

Owing to the practical aspects of computer science as well as the inadequacy of the "traditional" lectures and the availability of hardware, more and more educators are examining the emerging idea of "closed laboratories" which are often used in the teaching of physics as a method of teaching computer science. Indeed, 53% of computer science instructors in 4-year programs favored more supervised laboratories (closed labs) with computer science students as in the "physics model" (Dey & Mand, 1992, p. 13). Only 12% of them remained happy with the "mathematics model" which has been dominant in many computer science departments.

"Closed labs" are scheduled and supervised laboratory learning experience. Students are captive as in lectures, and they are expected to perform some tasks in the lab. Lab reports may also be required. On the other hand, "open labs" are unscheduled and unsupervised. Until recently, they were just called programming assignments.

Physics, chemistry, and most engineering fields have been running their laboratories for decades to provide hands-on experience, to promote critical observation skills, to encourage interactions among students in a controlled environment, to get familiar with equipment similar to those in the real world, and ultimately to enhance learning.

Computer science, on the other hand, is a very young field which goes through major changes almost annually. However, one trend that most computing educators agree on is the increase utilization of laboratories.

Some of the pioneers who utilize labs in computing believe that they are not achieving the full potential of laboratory experience:

> Lab assignments are not designed to allow students to discover important principles of computing. Thus, students receive training in program implementation rather than in the process of experimentation, discovery and evaluation which is more typical of advanced work in computing. (Tucker & Garnick, 1991, p. 46)

Though more and more schools have incorporated their introductory course with a laboratory component, little is known about how learning is enhanced by this classroom component. However, Thweatt pointed out that closed lab "make a positive difference" in examination scores (1994, pp. 80-82).

A new paradigm is emerging for education practitioners in computer science knowledge and skills (Denning, 1992, p. 83). The current model of education can be criticized because it treats "learning as acquisition of knowledge, and as an individual process" (p. 85). The "shifts in clearing

of education" (p. 85) should treat learning as a social process and competence should be demonstrated in action (p. 85). Instructors should not just be presenters or providers of instructional services, they can be coaches, guides, and facilitator (Brookfield, 1986, pp. 123-146; Denning, 1992, pp. 86-89). In fact, new approaches such as breadth first (Paxton, Ross, & Starkey, 1994, pp. 1-5), and software engineering (Leonard, 1991, p. 23) are being tried in various settings.

Montana Tech's Computer Science Program

Montana Tech is a small engineering college with a good regional and international reputation especially in mining and petroleum engineering. The use of laboratory in a programming course is consistent with Montana Tech's pragmatic hands-on approach in its other engineering programs.

The Computer Science Department at Montana Tech was founded in 1980 by a group of mathematicians with little or no industrial and computing experience. It was modeled after other computer science departments in that era. The design of the program reflects an assumption in the department was that anyone with a doctorate in mathematics can teach

anything. The situation at Montana Tech is far from unique. Most computer science departments had and still have strong ties with their local mathematics departments.

The computer science curriculum at Montana Tech was loosely based on Curriculum 78 (ACM, 1978). It was like a science degree with two emphases; one was in computer science and the other was in mathematics. Out of the 13 faculty members in the department of mathematics and computer science, 11 were mathematicians, 1 was a computer scientist, and 1 was an engineer. Owing to the necessity of getting accreditation by Computer Science Accreditation Board (CSAB), two major over-hauls have been done since 1993. One of the major changes was the introduction of computer laboratories in programming classes.

The CS 1 Course at Montana Tech

CS 210 Introduction to Computer Science I at Montana Tech is equivalent to CS 1 described in Curriculum 78 (pp. 60-63). CS 1 at Montana Tech has been a traditional three-credit course that had three 50-minute lectures a week traditionally at Montana Tech. It has been modified to two 50-minute lectures plus the 3-hour closed laboratories each week. There are other major modifications to the computer

science program at Montana Tech unrelated to this lab concept.

The first language taught has been Pascal since the birth of the computer science department at Montana Tech in 1981. However, this has been changed to the language C. This language chosen for Tech's CS 1 courses because the language C is the language of choice in the "real-world." It started to be the first language introduced to students in the Autumn of 1992. It is by no means the most teachable language, nor the most popular language for CS 1. Only 14% of 4-year colleges cover the language C at all (Dey & Mand, 1992, p. 11). In addition, less than 1% of the universities and 4-year college surveyed use C as their introduction language (p. 10). Tech is in a unique situation.

Although the lab idea was first introduced by the Association for Computing Machinery in 1979, the implementation of the concept did not catch on due to reasons like availability of machines and to some extent the tie with mathematics. Thus, the practice of laboratory experimentation is relatively new, and there are not many lab books on the market. Currently, there are no studies in published form on what works and what does not from students' point of view. Though Curricula 91 (ACM/IEEE-CS, 1991)

suggests that the use if closed lab is a good idea, it falls short of providing a concrete guideline. Some scientists even go as far as suggesting that computer science is not a science but an engineering discipline.

Although about 45% of the CS 1 students at Montana Tech are computer science majors, less than 60% of them move on to CS 2. Eventually, only 28% of the original group graduate each year. Considering that about the same number of students transfer into and out of the computer science program every year, the retention rate in computer science is extremely low. In CS 1, approximately over 40% of students are lost, and from CS 2 to their second year, 40% of the remaining students drop out of the program.

The New Laboratory in CS 1

Because of the trends in the field and in order to address this retention problem, closed labs were introduced as a requirement for CS 1 in the Autumn of 1996. Each lab session is 3 hour long. Lab activities have been used at Montana Tech in the computer literacy course to demonstrate the use of computer packages such as Microsoft Word and Excel. Laboratory activities in CS 1 included experimenting with short programs written for students to execute in the

lab as well as programs that students will produce. The activities can be divided into three stages of cognitive processes of (a) Record and Explain where students will record the results and report any expected and unexpected phenomenon, (b) Experiment and Discover where students are asked to modify working or semi-working programs to investigate concepts learned in the lectures, and (c) Design and Justify where students are also asked to design algorithms from scratch. Students are required to include design, analysis, implementation, and testing in their reports. All 3 activities require a write-up. The write-ups are design for students to demonstrate the application of concepts they learn in the lab by reflection. The write-ups are done within the 3 hour lab period. All documents, including design, programs, and write-ups, are turned in at the end of each lab.

There are usually about 160 students in CS 1. They are divided into two to three sections in both the lectures and the laboratories. With around 55 students in each lab., 30 machines are needed to anticipate the inevitable "down-time" even if students are organized in learning teams. The laboratory is set up in a room with 30 personal computers arranged in two circles. The two circles are layered in

which the inner circle has its computers on regular tables and the outer circle has its computers on drafting tables which are higher than the standard tables. The arrangement is designed for easy observation by the instructor. By standing in the center of the circles, the instructor can watch all screens without moving or disturbing the students. By walking around the outside circle, the instructor can observe all other laboratory activities like interactions between partners and can look at the frustrations on students' faces.

Problem

Computer science as a discipline has always been based on the "mathematics model." However, the computing field has matured sufficiently to have its own model as a separate discipline. There are positive changes emerging as computer science develops its own paradigm, and one of those being promoted is "closed labs." It is assumed that the added contact-hours in a structured laboratory setting will benefit students by leading to better learning. Montana Tech has implemented closed labs in its introductory computer science program. However, little is known about how students actually learn in the new setting.

Purpose

The purpose of this study was to describe how students learn problem solving skills and the syntax and the semantics of the language C in a closed computer laboratory.

Research Questions

The following general questions were used to explore how students learn in a closed computer laboratory:

1. How does the transfer of concepts from lectures to labs take place?
2. What learning strategies do students use to learn syntax and semantics in the language C, and what are the perceptions of the result?
3. What are the students' attitudes toward computer science as a result of the lab experience?
4. Why does the lab work or fail from the students' perspective?

Specific questions (see p. 74) were used in the two rounds of interviews.

Significance of Study

There are several areas that need insights for future modifications of the newly developed "closed lab."

Firstly, as more instructors get on the laboratory bandwagon, it is vital to learn how students interact in a programming lab. Such information will be helpful for forming teams of students in the future. If working in teams hinders learning, then this strategy should be avoided and replaced by working individually. On the other hand, if a collaborative learning team proves to be beneficial, the investigation should go deeper to explore the advantages and disadvantages of grouping students in different ways such as with similar or different levels of expertise, with different ages, with different gender, with different majors.

Secondly, the worthiness of the added contact hours needs to be explored. At present, microcomputer labs are used by a wide variety of courses at Montana Tech such as freshmen writing and mine modeling. With the increasing demands in the use of microcomputer labs, knowledge is needed on the learning process in these labs so that informed discussion can be made concerning the alleviation of this expensive resource.

Thirdly, it is important for the computer science department to know how different activities may affect the learning of different programming topics. For instance, if the technique "experiment and discover" takes too much time

for the average student, it is critical to find out if the technique "record and explain" is adequate in helping students to better learn about concepts like *arrays*.

Fourthly, it is also important to know the students' perceptions of the structured and predetermined activities. As suggested by ACM (1989), programming is only part of computer science; the lab component can only be successful if students find it engaging. Determining students' attitudes toward the lab can reflect if the designed activities are captivating.

Finally, the overall impacts of "closed lab" should be investigated. Thus, insights related to how the learning process takes place in the closed labs can be gathered.

Definition of Terms

The following terms will be used in later chapters:

Algorithm: A precise, unambiguous, step-by-step method of doing a task in a finite amount of time. An algorithm should be language independent.

Closed Lab: A scheduled and supervised laboratory in which students are captive and expected to perform some programming related tasks.

A Compiler: A translation program that rewrites high level language instructions into binary instructions or machine code which are then ready for execution.

A C Compiler: A compiler that translates programs in the language C into the designated machine code.

Debugging: A process of finding and eliminating errors in a program.

Open Lab: An unscheduled and unsupervised laboratory. Until recently, an open lab was referred to as a programming assignment.

Problem solving with the computer: A process from formulating the algorithm to a computer program running successfully for the prescribed problem.

Programming Environment: It is the collection of tools used in the development of software. The collection may only consist of a file systems, a text editor, and a compiler (Sebesta, 1996, p. 3)

Pseudocode: It is a sequence of statements that are close to a programming language, but more English-like, and free of rigid syntax requirements.

Syntax of a programming language: A set of rules for forming valid instructions of the language.

Semantics: The meaning of statements in programming languages.

Running of a program: The machine successfully follows the instructions in the program.

Limitations

Participants in this study were chosen from 3 sections of freshmen introductory to computer science course. The language chosen was C. It is possible that students at other institutes may respond differently to the learning of another language in the lab.

Delimitations

Participants were chosen for this study for their major, age, gender, and initial experience in programming. Majors ranged from computer science to business. Ages ranged from 14 to 49.

Assumptions

CS 1 labs at Montana Tech are only taught in the fall semester and the summer semester. CS 1 lab is part of CS 1 the course, and it cannot be taken separately. The prerequisite of CS 1 is high school algebra. Since there are less than 15 students in CS 1 each of the last 4 summers,

observations were limited to the fall semester only.

Students' participation was voluntarily.

CHAPTER 2

BACKGROUND AND REVIEW OF LITERATURE

History of Computing

Things that compute or simple calculating machines have been around for millennia. The abacus has been used for over 4000 years and is still being used in some parts of the world. Other mechanical arithmetic or algorithmic devices have been seen throughout history (Kidwell & Crruzzi, 1994, pp. 13 - 23); for example, Pascaline was the first automatic mechanical calculator invented by Blaise Pascal in 1642. Yet, the first large-scale electronic computer ENIAC, Electronic Numerical Integrator And Computer, was introduced only 50 years ago.

The Birth of the Computer

Although they are computers, the abacus and other bead frames are all completely non-automatic (Moreau, 1984, p. 4). Right before the second World War, John Antanasoff, a professor at Iowa State University, and Clifford Berry, Antanasoff's assistant, tried to solve the tedious systems of equations with 29 unknowns and 29 equations. No human, no matter how focus, could accurately solve problems like

this over and over again. They attempted to design an electronic digital computer to do the task. Unfortunately, no one at that time figured out how to represent numbers in such a machine. Antanasoff eventually avoided the difficulties of electronically representing numbers in base 10. He picked a voltage 0-2.3 to represent 0 and a voltage 2.3 and above to represent 1. Using this system, Antanasoff and Berry built a prototype before 1940 and called it Antanasoff-Berry Computer (ABC). World War II pushed the need for calculating shell trajectories accurately for new weapons. Trajectories tables were produced by teams of women, who were called "computers," by performing the calculations by hand. Thus, the earliest definition of "computer" was "one who computes."

In 1944, the first general-purpose electronic digital computer, ENIAC (Electronic Numerator, Integrator, Analyzer and Computer) was finally introduced at the University of Pennsylvania (p. 35). ENIAC was not completed until 1946 and it could compute a trajectory in 20 seconds. A person needed 2 days for the same task. However, the machine cost $500,000 and required 6 full-time technician to keep it running. ENIAC operators set the machine to solve problems by plugging in cables and switches. In fact,

solving new problems at this era meant rewiring the machine (Kidwell & Ceruzzi, 1994, p. 64).

Until recently, the use of electronic computer has been very expensive. From ENIAC in 1946 through the first supercomputer by Cray in 1972, and to all the mainframes in the 1970's before the first commercial personal computer in 1976, electronic computers could be afforded only by a few. However, today the $1000 computer with a Pentium chip has far more computational power than the $500,000 ENIAC. Computing technologies in hardware have evolved beyond most people's imagination, and software evolves with hardware every step of the way.

Generations of Languages and Hardware

Computers of the 1940's and early 1950's used vacuum tubes and programming was done in machine language which consisted of a small set of instructions recognized by and executed on the intended machines. Machine language was and still is difficult to use because it is written in binary. In the binary language which uses the base 2 mathematical system, everything is represented in 1's and 0's. For example, adding the contents of 2 registers requires the following binary sequence: 0000 1111 0000 0000 0000 0100

0000 0111. By nature, it is the furthest language from the problems people try to solve among all languages. Thus, assembly languages were developed to enhance the communication between human and the computer. Assembly languages can be characterized by the use of mnemonic codes and symbolic addresses. Programs written in assembly language are translated to machine language by assemblers, and then the sequence of instructions can be executed by the machine. A sample of machine language and the corresponding assembly language is shown in Figure 1.

Figure 1. Machine and Assembly Languages.

Machine Language represented in hexadecimal	Assembly Language using mnemonics
03 08 0A BC	LOAD R8, PRICE
03 07 0A BE	LOAD R7, TAX
0F 00 08 07	ADD3 R0, R8, R7
1C 00 0B 01	STORE R0, TOTAL
FF 00 00 00	END

Both programs above simply add the sale tax to the price of any item. Such an application has undisputed use. However, both machine languages and assembly languages are cryptic. Even to experienced programmers, they can be difficult to understand. To make matters worse, they are also too closely related to the structure of the machine. Programs written in one machine or assembly language can only be executed on one particular machine, hence making portability impossible. "Programming methods in that era were the most time-consuming and costly road block to the growth of computing" (Backus, 1976, p. 128). Languages in this period are usually referred to as low-level languages. The costs of programming and debugging far exceeded the cost of running a program. These problems sparked the development of high-level languages.

High-level languages, on the other hand, provides English-like code. COBOL, Commercial and Business Oriented Language, and FORTRAN, FORmula TRANslation, were the first two widely used high-level languages since the late 1950's. Though high-level languages during this period were primitive by today's standard because of the lack of high-level data structures other than arrays, they paved the road for the evolution of programming languages (Knuth & Pardo,

1976, pp. 264-266). They are much easier to follow than the low-level languages. For example, a statement to accomplish the same task as in figure 1 in COBOL is "assign Total the value Price plus Tax"; in FORTRAN this could be accomplished by "Total = Price + Tax." In the more recent language C, it would be "Total = Price + Tax;" with the semicolon terminating the statement.

Words such as assign and value take on special meanings as the symbols like = and + in FORTRAN and C.

With the introduction of transistors in the late 1950's and the integrated circuits in the mid-1960's, computers were able to be made much smaller and cheaper (Moreau, 1984, pp. 89-92). However, it was the invention of a highly dense integrated circuits by Intel known as the 4004 chip in 1971 that caused the revolution in the computer industry to ensure the availability and the affordability of hardware. The major periods in the evolution of hardware are in Table 1 summarized (Impagliazzo and Nagin 1995, p. 27).

Software followed the lead of hardware. Scores of high-level languages came out to utilize the development of new hardware. Three in particular have profound impacts in the computing industry as well as computer science

education. BASIC, Pascal, and C all came out in the early

Table 1. Generations of Computer Hardware.

Generations	Time Period	Principal Events
0	1642 - 1945	Mechanical calculators
1	1945 -1955	Vacuum tubes
2	1955 - 1965	Transistors
3	1965 - 1971	Integrated circuits
4	1971 - Present	Computer Chips

1970's. BASIC, Beginners All-purpose Symbolic Instruction Code, is perhaps the most popular computer language in terms of the number of users. It is a common first language introduced to students learning computer programming in high school. Pascal is the most popular language used in beginning computer programming course throughout the world

(Levy, 1995, p. 21). C, on the other hand, is the language of choice by software engineers and programmers. Today, newer languages such as Ada, C++, and Java have been developed for the fast changing field of computing. Languages are related in such a way that older languages help shape newer ones (Sebesta, 1996, p. 37).

The C Programming Language

C is one of the most popular languages among programmers. Unfortunately, the awkward name of the language C is because it is the successor of a short-lived language B which was the successor of BCPL (Basic Combined Programming Language). The name C does not stand for anything. C was originally designed and implemented by Dennis Ritchie in 1972 for the DEC (Digital Equipment Corporation) PDP-11 computer. Thus, C is not a new language by any means.

"C is a general-purpose programming language which features economy of expression, modern control flow and data structures, and a rich set of operators." (Kernighan & Ritchie, 1988, p. xi). C was strongly tied with the Unix operating system because both Kernighan and his colleague Ritchie worked on both Unix and C at Bell Laboratories. It

is a concise language with only 30 keywords. It is flexible, powerful, and well-suited for programming at any level of abstraction (Friedman, 1991, pp. 374 -375). Since the 70's, it has been well-received in many environments from personal computers to super computers. Combined with the smallness of the language (Kernighan & Ricchie, p. xi) and the standardization of C by Amercian Nation Standard Institute (ANSI) made C the most portable computer programming language. Essentially, C can be used on any machine because C compilers are available on any platform from a microcomputer using the Pentium chip to the most powerful Cray supercomputer.

Although C is a high level language, it is so expressive and efficient that it replaced assembly language in many circumstances. As a result, C is the language of choice for most software engineers and programmers and it is perhaps the most dominant language in the field of computing.

On the other hand, C is the not easiest language to learn. It is also not the best language for beginners because its flexibility sometimes causes unexpected results that could be confusing especially for beginners. Some statements in C could be confusing, redundant, and

frustrating to students. To simply add one to a variable X, there are several statements that can accomplish the job:

1. `X = X + 1;`
2. `X += 1;`
3. `X++;`
4. `++X;`
5. `fun(&X);` *provided that function fun use one of the above 4 statements to add 1 to X.*

To make matters worse, when statements 3 and 4 are mixed within other statements, there could be undesirable side effects. For example, they could create

```
X = 2;
Y = ++X;
```

does not equal to

```
X = 2;
Y = X++;
```

In the first case, both X and Y become 3. In the second case, X becomes 3, and Y remains 2. The position of the ++ decides when the increment of X occurs. There are numerous

other occasions that make C not the best choice as a first language. Consequently, very few colleges use C as their first language (Dey & Mand, 1992, p. 10).

Computing as a Discipline

As the information technology becomes more and more important socially and economically in every community, educators have to update the evolving computing discipline to match the changing needs (Shaw, 1991, p. 9). "Computation is joining the scientific paradigms of experimentation and theory" (p. 17). Computing courses are now required in almost every major in college. Thus, changes in the computing curriculum affect majors and non-majors alike.

Even with all the changes, lab and programming will be essential parts of the computing curriculum. While lectures tend to concentrate on theoretical and abstraction processes, the labs can help students learn and practice the design, implementation and testing of software. Since programming languages are regarded as tools for computing professionals, to teach the use of a tool with hands-on experiments in the labs seems logical. Some may go as far as dropping the lectures all together and teach programming

in the labs entirely (Bruce, 1991, p. 30). No matter how far one goes, labs will remain the fact of life in the field of computer education for years to come.

Whether a course's concentration is software packages (e.g. Microsoft Office), programming, or a breadth-first approach such described by Paxton, Ross, and Starkey (1993, pp. 68-72), the lab component is inevitable. For software developers or traditional computer scientists, Association for Computing Machinery (1991) recommended 10 subject areas in computer science: (a) Algorithms and Data Structures; (b) Architecture; (c) Artificial Intelligence and Robotics; (d) Database and Information Retrieval; (e) Human-Computer Communication; (f) Numerical and Symbolic Computation; (g) Operating Systems; (h) Programming Languages; (i) Software Methodology and Engineering; (j) Social, Ethical, and Professional Issues.

For non-majors such as architecture, business, chemistry, education, and engineering, mere programming skills may no longer suffice for their specialty. Nevertheless, taking an introductory course in computer science opens the possibility for those majors to appreciate areas of computer science such as artificial intelligence, data communications, or graphics. As a matter of fact,

upper division computer science courses are constantly taken by non-majors.

Laboratory Activities

Laboratories are used to support the learning process by offering students well-chosen, short, well-paced exercise (Hartel & Hertzberger, 1995, p. 15). In an introductory programming course, laboratory activities can be divided into the three major categories: (a) Record and Explain; (b) Experiment and Discover; and (c) Design and Justify.

Record and Explain. One of the goals of the lab is to make the programming concepts studied "operational and allow students to ascertain that the material is understood" (p. 15). To achieve such an objective, students are given working programs to run. The results of the run are recorded. Follow-up questions are then answered.

This activity can demonstrate the students' proficiency in the use of the programming environment, such as with the editor, the compiler, or the network set-up in the lab. The follow-up questions are designed to test if the students can relate a particular concept to the program. Step 1 of the lab in Appendix C demonstrates

the process of "recording" the result of a program given to students. All they have to do is to extract the program from a network drive set-up by the instructor. In other words, they do not even need to type in the program. Once the results are recorded, they are asked to relate the result to a concept related to simple logic. The result of the program is illustrated in Figure 4.

Figure 2. The Result of the Program Logical And.

Truth table of logical operation && (and)

operand 1	operand 2	operand 1 && operand 2
0	0	0
0	1	0
1	0	0
1	1	1

Students learn from lectures and the text book that 0 means No and that 1 means Yes. In this example, they are supposed to relate the result of the program to getting a "ham and cheese" sandwich for the instructor. The example

is a classic way to explain the logical "and" operation. If a "ham and cheese" sandwich is ordered, they can bring back four different kinds of sandwich:

1. A sandwich with no ham and no cheese, (e.g. a roast beef and bacon sandwich).
2. A sandwich with no ham and only cheese, (e.g. a roast beef and cheese).
3. A sandwich with ham but no cheese, (e.g. a ham sandwich, or a ham and lettuce sandwich).
4. A sandwich with both ham and cheese.

The object of the lesson is for students to realize that the four scenarios above correspond to the truth table. If a ham and cheese sandwich is ordered, the outcomes can be summarized in figure 5.

Figure 3. Scenarios of a Ham and Cheese Sandwich.

Ham	Cheese	Sandwich
No	No	No (not okay)
No	Yes	No (still not okay)
Yes	No	No (Still not okay)
Yes	Yes	Yes (okay)

Students are expected to relate the above activity to the logical "and" operation.

Experiment and Discover. Experiment and Discover activities encourage students to be adventurous. Once they are confident enough, they are required to experiment in order to complete the lab programs. Step 3 in Appendix C requires students to modify the working program in Step 1. In order to make it work, they must make 6 modifications. They must change all logical "and" or logical "or" (i.e. "&&" to "||" operators). Missing just one will not get the supposed result as in Figure 6.

Step 4 in the same lab requires even more experimenting to discover the apparent logic that "not ham or cheese" is the same as "no ham and no cheese." In the process of discovery, students should also realize the difficulties of printing tables in which all columns align perfectly.

Another typical discovery experiment is to encourage students to associate algorithms that they already know from real life to computing algorithms. Appendix D is a

lab from Shiflet's (1993, pp. 314-315) text. In this lab exercise, students are asked to guess a number within a

Figure 4. The Result of the Program Logical Or.

Truth table of logical operation || (or)

operand 1	operand 2	operand 1 || operand 2
0	0	0
0	1	0
1	0	0
1	1	1

predetermined range, such as from 0 to 1022 which is generated by a computer program.

First, they are asked to guess from 0, 1, 2, and so on sequentially in one step. The method is called sequential search in computer science.

Then, they are instructed to guess the middle of the range. The computer program keeps track of the number of guesses they needed before the number is found. After every guess, they are told to go higher or lower if the guess is wrong. This provides "instant feedback to the students" (Hartel & Hertzberger, 1995, p. 15). They are told to keep the "higher" half or "lower" half and repeat the same process until the number is found. This time the algorithm is appropriately called binary search. Most students actually learned both methods from past experience or from the TV show "The Price is Right."

When the range is from 0 to 9, there is little difference between the performance of the two methods. However, after they change the range to 0 to 1022, they realize the superior efficiency of the binary search. In the lab exercise following the experimentation, they are asked to analyze the two algorithms mathematically. If they cannot perform the analysis, at least the exercise stimulates experimentation and raises questions for further discussions (p. 15).

Design and Justify. The two activities above are excellent learning approaches in programming.

Nevertheless, they both lack the promotion of creativity. Part of programming requires inventiveness within the boundary of the language syntax and semantics. To foster such learning, students must demonstrate the ability to present solutions in an acceptable manner. Flow-charts, mpl's (Model Programming Language), and pseudo-codes have been used by programmers to describe the solution of a programming problem since the first program was written in 1940's. Thus, design and justify activities allow students "to concentrate on programming rather than the distracting details" (Starkey & Ross, 1984, p. xx) of a programming language.

A systematic approach to problem solving that involves programming demands four basic steps of analysis, design, implementation, and testing (Shiflet, 1995, p. 112). Design and justify activities in the lab are used to "offer the student the opportunity to discover solutions to problems" (Hartel & Hertzberger, 1995, p. 15). Instant feedback can be provided to guide students to one of the solutions as well as to promote creativity to problem solving (p.15). The justify part helps students to debug the design of a program to ensure the program logic

satisfies the requirements, is sound, and covers all possibilities (Koffman, 1989, p. 95-96)

Appendix E is an example of "design and justify." Students are asked to design an algorithm to do the simple task of figuring out all possible tickets in a lottery in which 3 balls or numbers are drawn from 10. Students are given the opportunity to discover or design the solutions to the problem (Prather, 1992, p. 61).

Computers and Cognition

Technology and Education

Before humans invented reading and writing, pictures and gestures were used to convey ideas (information). Pictures later became ideograms. Gestures became sign language. Educators have been using technology to enhance teaching and learning for years. Technological changes have taken many forms, which included the movement from an overhead projector to a computerized grade book system, from radio to television, and from personal computer to the Internet, "We have new tools for learning and teaching which change how our minds work" (White, 1988, p. 6).

In 1813, Thomas Jefferson envisioned that "books will soon be obsolete in the school" (quoted by Cuban, 1986, p.

11). Today, the whole library of Congress' English-language holdings can be stored on three 4.75-inch compact discs. CD-ROM (Compact Disc Read Only Memory) provides the ability to give teachers and students random access to thousands of visuals in milli-seconds. It also allows users to explore enormous amount of textual data. As Mageau (1990) predicted, today digitized motion video is almost as common as video tapes (p. 28).

Today, computer networks provide educational institutions electronic mail systems as well as the abilities to share information and resources through various wide area networks. The prophecy that "telecommunications one day soon may become an indispensable learning tool in U.S. classrooms" (p. 29) is already a reality.

Multi-media is the one of the few new lingo that perfectly describes its meaning. Multi-media provides sensational stimulants that integrate text, audio, graphics, still images and moving pictures into a single, computer-controlled product. Together with desk-top publishing, reading definitely is taking on a whole new meaning.

Computers in Education

The earliest electronic computers were installed in colleges approximately 50 years ago. They were initially used to solve multiple unknowns in equations or perform other intensive calculations. Since John Kennedy developed the BASIC language at Dartmouth College, computers and higher education became even more inseparable. Learning about the use of the micro-computer and its software packages is required in almost every college degree program. Computers can even replace physical instruments in a chemistry lab (Ivey, 1992, pp. 4-8). Simulation programs can be used to conduct experiments that may be too dangerous or expensive to perform (Parker, 1996, p. 14-17). As computers get into every facet of life, they remain instrumental in education.

Computer Assisted Instruction (CAI) helps students learn at their own pace. The drill and practice set up in the "remedial mode" is especially helpful in high school algebra if the purpose is to help student increase math scores in their Scholastic Aptitude Test (SAT). Though adult educators might argue that students using CAI are too passive in directing their own learning (Knowles, 1980, p. 48), the potenital of computers in education is only in its

early stages (Parker, 1996, p. 7).

Adult Learners in Computer Science

"Changing demographics is a social reality shaping the provision of learning in contemporary American society." (Merriam & Caffarella, 1991, p. 6) America is becoming a nation of adults. It is estimated that by the year 2000, the largest age group will be 30 to 44 year olds (Cross 1981, p. 3). With the large age group in their so-called "most productivity years" and with the average American making between 5 to 10 job changes in a lifetime, continuing professional education is becoming more and more critical. In order to be competitive in the world-wide market, re-training of the United States workforce is inevitable. "Lifelong learning is essential to professional productivity, individual potential, and international competitiveness" (Anderson 1991, p. 17). More and more college students have adult responsibilities. Many of them have a full-time job and take classes after work. Some adults enroll in the computer science program to pursuit their first degree. Some return to college to retrain themselves in a field which changes as fast as one's imagination. Other adults return after losing their

job; as in the computer science program at Montana Tech, this included petroleum engineers and high school teachers. "Lifelong learning is not a privilege or a right; it is simply a necessity for anyone, young or old, who must live with the escalating pace of change: in family, on the job, in the community, and in the world-wide society" (Cross 1981, p. ix).

The computing experience that adults bring to the learning situation sometimes is the opposite of the common wisdom. The "andragogical model assumes that adults enter into an educational activity with both a greater volume and a different quality of experience from youth" (Knowles, 1980, p. 10). Computer science is usually the opposite. For example, in an infromal study at Montana Tech (see Appendix A), the two youngest students (age 14 and 16) in a C class had far more experience than the two oldest (age 44 and 45). This is actually quite a typical phenomenon in computer science classes.

Adults are motivated to learn after they experience a need in their life situation, and they enter an educational activity with life-centered, task-centered, or problem-centered orientation to learning (Knowles, 1980, pp. 78-95). Thus, adult learning activities should be organized

around their needs regardless of the subject. Both teachers and students share the responsibility of learning, course development, and even outcome evaluation. With more and more adults in the field of computing, adult learners provide instructors a new challenge.

Educational Objectives

To design effective learning activities, one must explore the learning sequence students use in the learning process. The cognitive domain presented in Bloom's taxonomy (1956) can be summarized in 6 levels. Steinaker (1975, p. 15) identified 5 levels in the experiential domain which matches the lab activities in the computer science laboratory.

Figure 5. Bloom's Classification of Educational Objectives.

Level	Cognitive Domain
I	Knowledge
II	Comprehension
III	Application
IV	Analysis
V	Synthesis
VI	Evaluation

Figure 6. Steinaker's Classifications of the Experiential Domain.

Level	Experiential Domain
1	Exposure (Comprehension)
2	Participation (Application)
3	Identification (Involvement)
4	Internalization (Adoption)
5	Dissemination (Commitment)

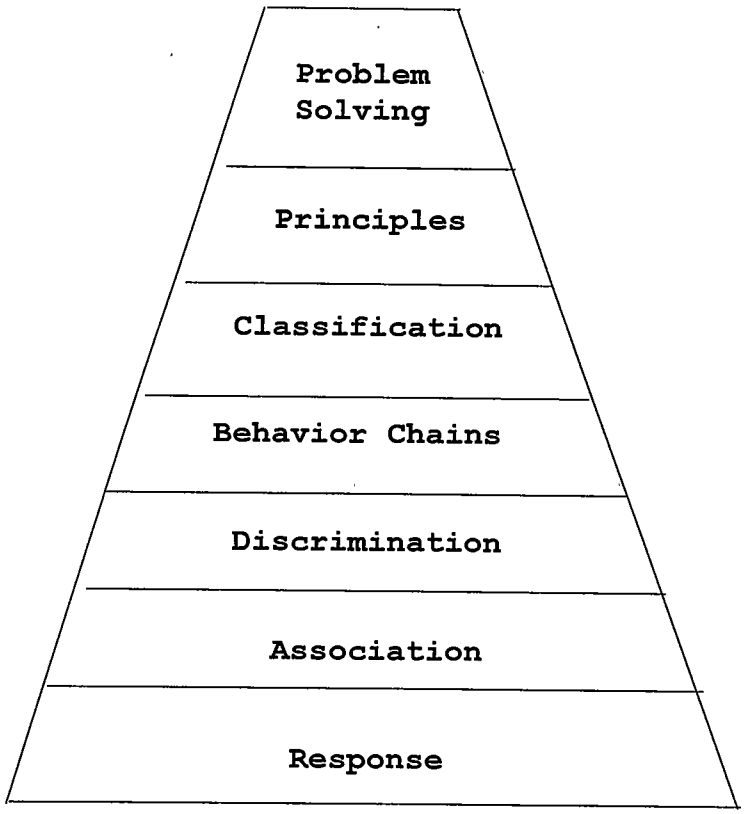
The typical classroom set-up with the traditional one-way transmission (Knowles, 1984, p. 15), most learners could hardly reach level 3. On the other hand, learning the syntax of a programming language can be viewed as knowledge based process (Winograd, 1983, pp. 2-29) which may only involve levels 1, 2, and 3.

In order to design a solution of a problem, students must be asked to interact with situations that are realistic, open-ended, complex, and largely un-structured. This encourages students to disseminate different concepts into a computer program. It requires the application of principles, the very top of Gagne's learning hierarchy. (1965)

The partnering in the computer laboratory can be loosely coupled or as elaborate than cooperative learning (Brown & Palincsar, 1989, pp. 397-408). Student-student interaction can be structured in three ways: competitively, individualistically, and cooperatively (Johnson, Johnson, & Smith, 1991, p. 2). Students should not be graded against one another on a norm-referenced basis and should be encouraged to work together to accomplished shared goals in the lab (pp. 2-3). The process then encourages interpersonal communications. Thus, besides problem solving skills, laboratory can enhance a large inventory of skills and attributes that are valued in computing education as by-products: communicative, organizational, leadership, and planning skills on top of the assumed computational (programming) skills (Harrisberger, Heydinger, Seeley, & Talburtt, 1976, pp. 3-14).

The teacher's role in the laboratory learning changes quite naturally from "instructor" to "supervisor/consultant" or "facilitator of learning" (Knowles, 1984, p. 14). Only demonstrations of technical material should be given in the early part of each laboratory period (Tucker, Bernat, Bradley, Cupper, & Scragg, 1995, p. x).

Figure 7. Gagne's Learning Hierarchy.



Model of Instructions: Andragogy and Pedagogy

Andragogy is derived from the Greek word aner which means man, and pedagogy is derived from paid which means child (Darkenwald & Merriam, 1982, p.13). Both word refer to the art and science of helping learners' learn, and the learners just differ in age or "the self concept of being responsible for one's own life" (Knowles, 1984, p. 9).

The pedagogical model has dominated all of education since school started being organized in the seventh century (p. 8). The andragogical model has been one of the theories to unify the field of adult education (Merriam & Caffarella, 1991, p. 249). The difference in their assumptions can be summarized in Table 2 (Knowles, 1984, pp. 8-12).

Table 2. Assumptions of the Andragogical and Pedagogical models.

Learner	Pedagogical model	Andragogical model
self-concept	dependent	self-directed
experience	little value	great source for learning
readiness	age	need to know
orientation	subject-centered	task/problem centered
motivation	external	mostly internal

There are obvious implications for the program design of the 2 models. The basic format of the pedagogical model is content plan while in the adragogical model is process

design (pp. 13-14). The difference can be summarized in Table 3 (pp. 13-20).

Table 3. Program Design for the Pedagogical model and the Andragogical model.

Element	Pedagogical model	Andragogical model
climate setting	competitve, formal	collabrative, informal
planning	by teacher	by both facilitator and learner
diagonsing need	by teacher	by both facilitator and learner
learning objectives	by teacher	by both facilitator and learner
Sequence of learning	by teacher	by individual learner
evaluation	by teacher	by both facilitator and learner

Though Knowles suggested that pedagogy should be replaced by andragogy for both children and adults (Knowles, 1978, p. 53) and the 2 models seem to stand at

the 2 opposite corners, "the pedagogical and andragogical model as parallel, not antithetical" (Knowles, 1984, p. 12). Thus, the 2 models can work together as the continuity of human development.

Laboratory Learning

"Learning is a term with more meanings than there are theories" (Brown & Palincsar, 1989, p. 394). Various educators view learning differently. For example, Horton (Adam, 1975, pp. 205 - 206; Moyers, 1990) and Freire (1973) relate learning to social movements and changes. Socrates (Grube, 1976, pp. 1-32), on the other hand, asked students a series of questions so that they could search within themselves. To Dewey (1938), learning was fundamental to growth and democracy. Skinner (1974) maintained that learning is crucial for a species and its survival (pp. 205). Maslow (1954) argued that it is the process of self-actualization (pp. 203-208) and Rogers (1996) felt it promotes fully-functional individuals (p. 288). Despite the many views concerning learning, many of those teaching in computer science laboratories support Gagne's (1965) view of learning as "problem solving" by applying programming principles in the computer laboratory.

Experiential learning or learning by doing has been a fundamental concept in education for centuries. The master-apprentice approach has been utilized since the time of the Greeks. The laboratory has always been regarded as a necessary component of the educational process. In basic sciences, laboratory exercises are as old as the fields themselves. Though computer science is a very young field, there is an increasing emphasis in computer science education on hands-on programming exercises and internships before graduation. In computer science, project activities are common among old and new curricula. Laboratory learning, a subset of experiential learning (Knowles, 1984, pp. 417-420), is just a project activity conducted in smaller scale but a more structured manner.

The use of the laboratory as an instructional method was first introduced in ACM Curriculum 78 (p. 63). It has, however, not been fully exploited in practice even though other fields have shown clear benefits in terms of learning.

CHAPTER 3

METHODOLOGY

This descriptive study utilized a naturalistic case study design. Both quantitative and qualitative data were collected to obtain information from participants in the beginning and at the end of the study. Interviews with students were conducted in the middle of the Fall semester of 1996 and again at the end of the same semester.

Naturalistic Inquiry

Though the application of naturalistic inquiry may be relatively new to educational research, it has been used by anthropologists for years. As a matter of fact, it is often referred to as the anthropological approach. Its recent growth in both interests and acceptance is quite natural and good for educational research.

Until the recent emergence of the theory of chaos, mathematicians and physicists tended to think that there exists a formula or a set of formulae to describe or predict any phenomenon (Gleick, 1984, pp. 7-8). If the description or prediction is not exact as in weather prediction, the problem must stem from the fallacy of the formulae. The quest for one truth characterizes the tradition in which rationalistic inquiry was formed. The basic belief of

rationalistic inquiry is the fact that there is one objective reality. A rationalistic inquirer's job is to uncover the truth which can be described mathematically, and thus the situation can be predicted and replicated (Huck, Comer, & Bonds, 1974, p. 11).

Rationalistic inquirers have been called logical positivists who seek facts and causes of social phenomenon with little regard for the status of individuals being studied. To a rationalistic inquirer, research is an objective quest for replicable findings (pp. 369-371). The purpose of the research is to test a hypothesis or verify a theory in order to generalize or to infer (McClave, 1986, pp. 2-4).

A naturalistic inquiry(NI) researcher, on the other hand, is interested in describing and understanding a phenomenon from the subject's own frame of reference. NI researchers believe there exist multiple realities. Thus, NI researchers are sometimes called phenomenologists. The main purpose of NI is the discovery of phenomena. (Bogdan & Taylor, 1975, Chapter 1)

The setting in which research is performed differs between naturalistic inquiry and the rationalistic inquiry (RI) paradigms. RI is best achieved in a laboratory setting or behind the "non-existing one-way glass" to insure

objectivity. RI considers the world as composed of variables. By manipulating the predefined independent variables, the researcher investigates the effects on dependent variables which is predefined. Thus, in a typical rationalistic inquiry, the researcher identifies all independent and dependent variables of interest and then randomizes the selection of samples in order to measure the effects and to reach a conclusion. The key is to use laboratory control if possible. When laboratory control is not possible, statistical manipulation is employed.

Since NI is more concerned with description or understanding of phenomena, checking of the discovery is done through "triangulation" in which one source is tested against another until the researcher is satisfied that the interpretation is valid (Guba, 1978, p. 13). Thus,

> It is important to provide multiple data source and methods of collection. It is also important to describe techniques that were used to check and validate analyses as the research proceeded. (Owen, 1982, p. 13)

Owing to the difference in philosophy in the two paradigms, data collection techniques are approached differently. RI tends to favor survey instruments. Random sampling is preferred. The approach is structured. The design is fixed. (Devore & Peck, 1986, pp. 233-246)

NI researchers use interviews and observations to collect data. The sampling technique is that of purposeful sampling. The approach is exploratory. The design is flexible or at least incomplete because "the design emerges as the investigation proceeds" and "it is in constant flux as new information is gained and new insight is formed" (Guba, 1978, pp. 13-14).

At the end of a rationalistic study, there is usually a detailed report with figures and charts. The hypothesis is either affirmed or disproved based upon the data gathered and analyzed statistically (Huck, Comer, & Bonds, 1974, p. 365). The effect of independent variables on dependent variables is discussed. Depending on the statistical method chosen, confidence-level and confidence interval may be included. With the help of a computer, complicated statistical relationships can easily be obtained.

In naturalistic inquiry, relevant information which has been collected is described. Research descriptions tend to be "thick and rich" because they are filled with quotes, anecdotes, and personal stories. The NI researcher often consider gestures, language, and behavioral patterns of the subjects as significant descriptive data (Guba, 1978, p. 7). The investigator may then conceptualize issues by deriving categories that fit the information collected. While data

collection in a rationalistic study ends when the researcher has collected a predetermined amount of data, naturalistic studies do not have these clear deadlines. Instead, there are at least four ways which guide the researcher in terminating the collecting process of the research. Exhaustion of sources, saturation, emergence of regularity, and overextension are used to detect if there are no new situations, or same pieces of information are recurring, or the area feels integrated (Guba, 1978, pp. 60 - 61). This process seeks to reach an in-depth understanding of the situation. Though theories are not proved, they may emerge and be proposed for further exploration. While RI researchers are reductionists, NI researchers are expansionists.

Case Studies

A case study is an in-depth and systematic investigation of an individual, a group, an institution, a process, a social group, or a phenomenon (Gay, 1992, pp. 235-236; Merriam, 1988, p. 16). It "seeks holistic description and explanation" (Merriam, 1988, p. 16). In short, a case study examines an instance in action. It focuses on one particular phenomenon. It produces "thick" description of that phenomenon to illuminate the reader's

understanding of such phenomenon (pp. 11-13). Thus, "generalization, concepts, or hypotheses emerge from an examination of data" (p. 13).

Unlike the rationalistic inquiry in which variables are manipulated, the researcher needs little control in a case study. It is because the research questions in case studies are "how" and "why" (p. 9). Furthermore, the case study deals with "a bounded system" (Stake, 1988, p.255). The researcher, who observes behavior in its natural setting, is the primary instrument for data collection and analysis.

> The qualitative case study can be defined as intensive, holistic description and analysis of a single entity phenomenon, or social unit. Case studies are particularistic, descriptive, and heuristic and rely heavily on inductive reasoning in handling multiple data sources (Merriam, 1988, p. 16).

Thus, a qualitative case study uses all methods of data as diverse as testing and interviewing (p. 10) to report findings or reveal properties to discover new meaning and to extend a reader's understanding of a situation in a comprehensive and expansive report.

The naturalistic case study was chosen for this study because the lab set up in the computer science department at Montana Tech is new and unique and because the focus of the

study is to discover what works and why for students in this learning process and to explain the relationships between different factors. Moreover, this situation was unique because no one at Montana Tech had done lab learning like this study and similar situations are not reported in the literature. It is impossible to predict all independent variables and dependent variables as in a traditional rationalistic study without overlooking valuable unforeseen information. These labs are systems. The study sought to understand the experience of students in the newly designed labs in the introductory computer science course. Thus, a naturalistic case study was chosen. However, some traditional quantitative tools were used to gather student information to aid this study. This investigation was done by processes of cross-checking, triangulation, and re-cycling until convergence was achieved (Guba, 1978, p. 13).

Research Population

Participants in this research project were all 3 lab sections of students in the course CS 210 Introductory to Computer Science I. The course consists of two 1-hour lectures and a 3-hour lab. Most students come from several majors: Computer Science, Engineering Science, and Business. There were 155 students in the three sections. Of these,

134 students participated in the Beginning-of-semester Survey. The age range was from 14 to 45 with 53% younger than 20, 23% in the range from 21 to 24, and 24% older than 25. The mean age was 20.5, and the median was 19. There were 47% freshmen, 35% sophomores, 9% juniors, 9% seniors, and less than 1% graduate students. Most (84%) took the course because it was required. Almost 55%

Table 4. Profiles of Students.

Variable	Number of Students	Percentage
Major		
computer Science	54	40
engineering science	36	26
business	24	18
others	20	15
Total	134	99
Year in School		
freshman	63	47
sophomore	47	35
junior	12	9
senior	12	9
graduate	0	0

Table 4. Profiles of Students--Continued.

Variable	Number of Students	Percentage
Age		
14	1	.75
16	2	1.49
17	1	.75
18	32	23.88
19	36	26.86
20	10	7.46
21	10	7.46
22	2	1.49
23	5	3.73
24	7	5.22
25	4	2.98
27	5	3.73
28	8	5.97
30	1	.75
31	1	.75
32	2	1.49
33	1	.75
34	1	.75
38	1	.75
40	1	.75
44	2	1.49
45	1	.75

Table 4. Profiles of Students--Continued.

Variable	Number of Students	Percentage
Programming Experience		
Yes	72	54
Yes (in the language C)	12	9
No	62	46
Gender		
male	92	69
female	42	31

of the group had programming experience. However, only 9% had used C before taking the course. The gender ratio was more than 2 males to 1 female. Only 31% were female students. Surprisingly, only 3 (2%) students out of 134 surveyed had never used a computer before this course; 66 (49%) owned a personal computer.

Procedures

The laboratories in which data were gathered had been set up physically prior to this study. The Turbo C compiler had also been chosen for the lab although VAX C, Visual C++ and Borland C/C++ were also available in the lab. VAX C, Visual C++, and Borland C/C++ are programs to convert the programs written by the students in C to machine code so that

student programs can be executed and observed. Different compilers provide different programming environments in terms of editing and debugging (finding syntax and logic mistakes) programs.

The situation at Montana Tech was unique. First of all, even though the language C is an unpopular beginning language, it was the language used in this course because C was the most used language in the industry. Secondly, the ACM Curriculum Committee recommended a 2-hour lab for a 3-credit CS 1 course (ACM, 1979, p. 63), yet a 3-hour lab was chosen for Montana Tech's CS 1 and this study because students were asked to turn in their lab reports at the end of each lab instead of the following day. Since this study sought to discover how learning is enhanced in a computer lab, a qualitative case study (Merriam, 1988) with direct observation and systematic interviews was used.

Pre-study assessment (see Appendix A) was given to all students. Based on the pre-study survey and the first several lab observations, 21 students were chosen for in-depth interviews. At least two interviews per student were conducted during the fall semester. One was in the middle of the semester, and the other was at the end of the semester. Post observation assessment (see Appendix B) was also given to every student in the course at the end of the semester.

Observations

Students were observed in the labs throughout the Fall Semester 1996. Notes were taken extensively in every lab to record student behavior, how students interacted, unforeseen events, time needed for different activities, students' reactions towards labs, as well as student approaches to the different particular learning activities of Record and Explain, Experiment and Discover, Design and Justify. This data were related to their age, gender, major, and prior experience.

The physical setup of the labs was designed to help the researcher observe as well as student learn. Both the physical and programming environments are discussed in the following sections.

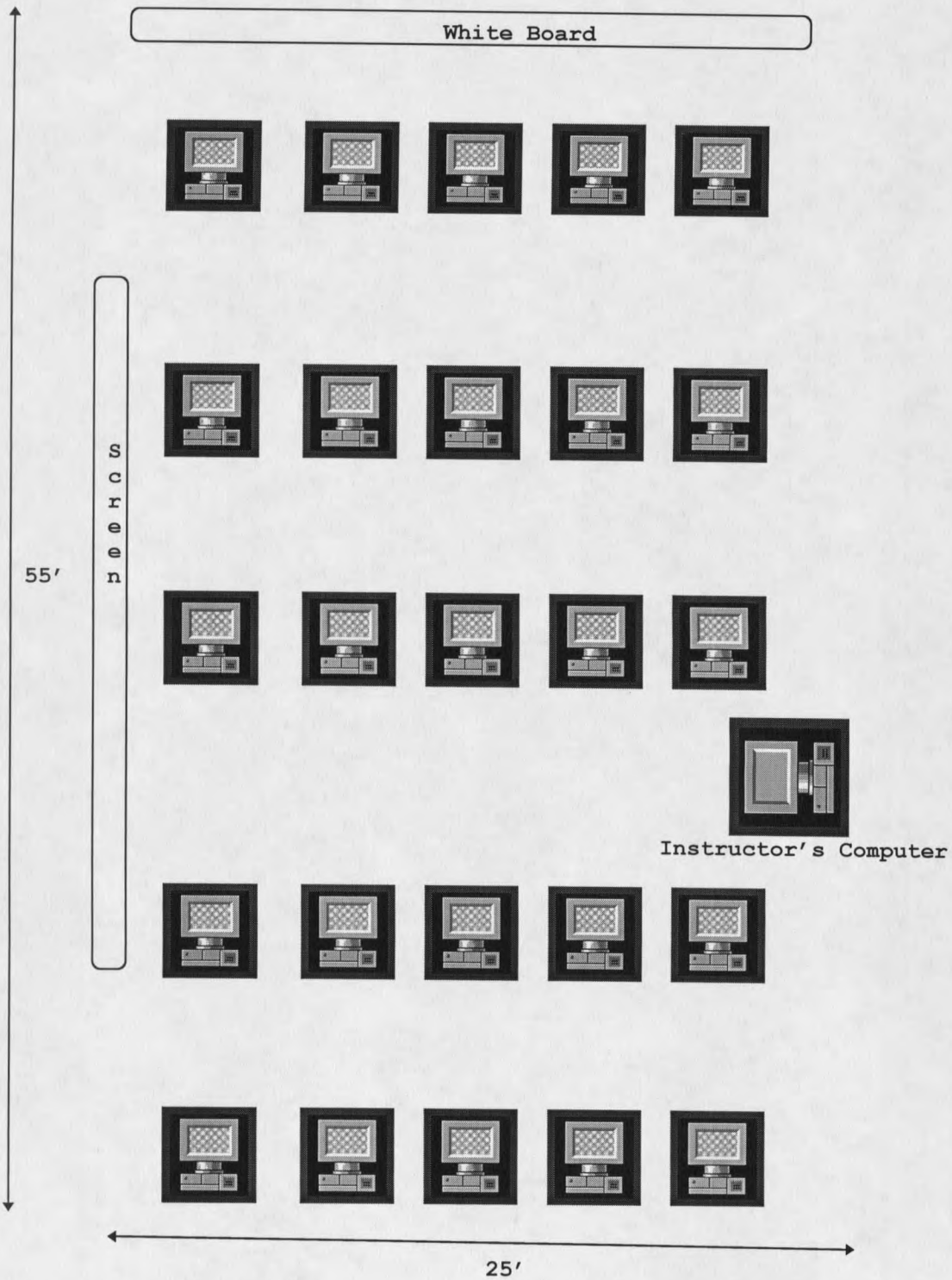
The Physical Environment. The physical environment can play an important part in enhancing learning (Knowles, 1984, p. 15; Merriam & Caffarella, 1991, pp. 31-32), and it can "facilitate the acquisition of content by the learners" (Knowles, 1984, p.14). The physical condition should be comfortable and conducive to interaction (Knowles, 1986, p. 7).

Montana Tech has two different types of labs: instructional labs and open labs. Instructional labs are

labs where classes are scheduled, and thus can be used in as "closed-labs". There were also microcomputer clusters for students to use as "open-labs." There were four instructional labs at Montana Tech. The four instructional labs were used by all departments for such courses as computer aided design for engineering departments and desktop publishing for communication courses. The four labs have different number of machines and physical configurations. Of the three lab sections required for CS 1, two labs were scheduled in a lab with 30 machines and a "circular setup." The other lab was scheduled in a lab with 25 machines and the traditional layout in which machines were on 5 long tables in 5 rows facing a white board. Each table, all of the same height, had five machines. All observations were done in all three lab sections. However, the observer could only observe one row at a time in the traditional set up. Figure 8 shows the lab with 25 machines. The lab is approximately 25 feet by 55 feet. This lab may be referred to a rectangular lab.

The other lab had machines setup in two sets of tables that formed three quarter circles. The inner circle had 10 machines. They were on small tables of the same height. The outer circle had 20 machines on drafting tables about

Figure 8. The Layout of the Rectangular Lab.

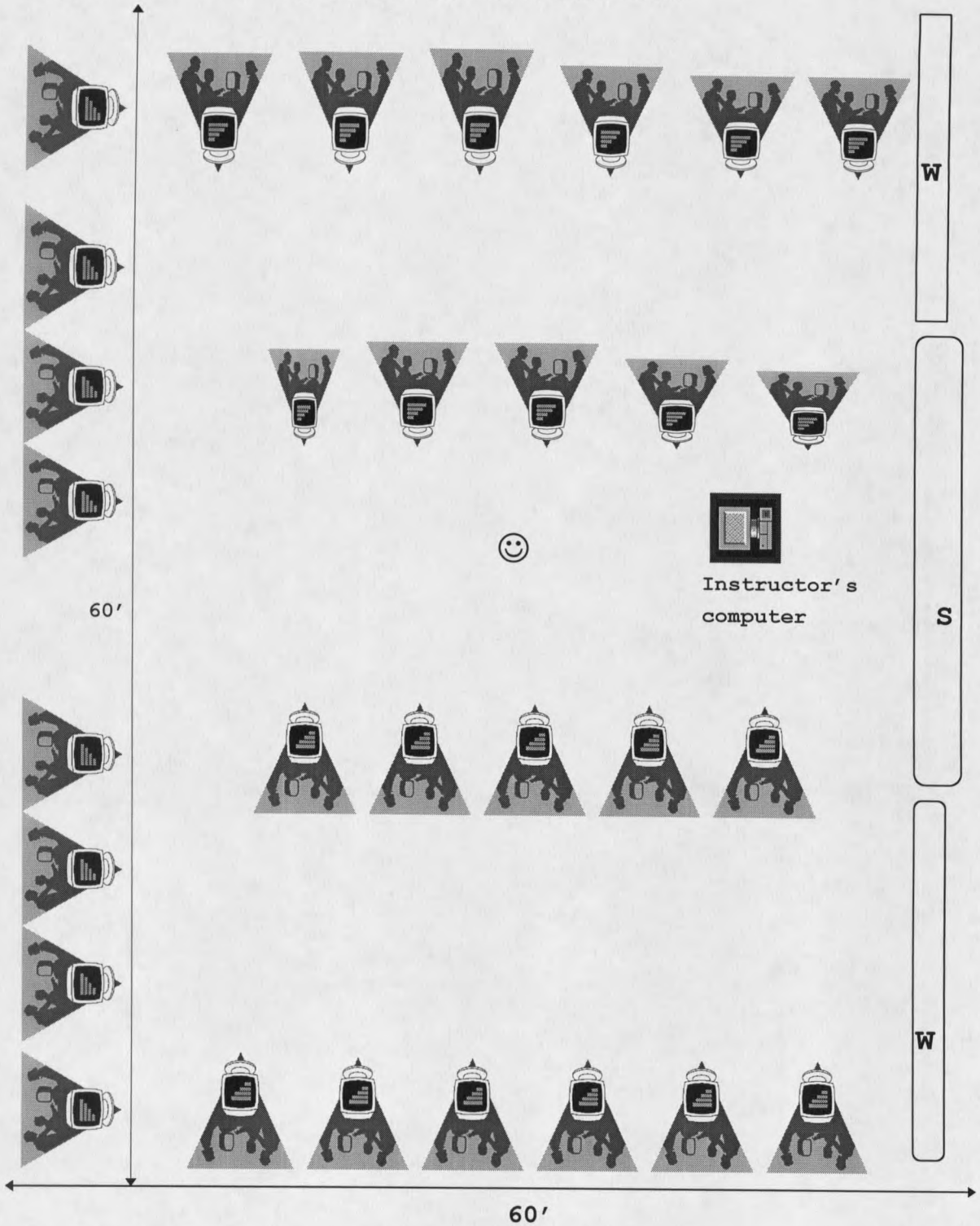


one foot taller than the tables in the inner circle. Thus, the observer could stand in the center and observe 15 machines all at once. Another crucial difference of the two labs is the size of the monitor. All monitors in the "circular" lab were large 17" monitors. All other labs at Montana Tech had only 14" monitors. Even though the details of each screen might be difficult to make out, students' physical activities and programming activities on the screens were easily observed on the 17" monitors. This lab is in a room which was approximately 60 feet by 60 feet. There was much space between tables and thus machines. This lab may be referred to as the circular lab.

Both labs also had a projection system to project the monitor of the instructor's machine to a large screen. In the circular lab, the instructor's monitor and projection system was located in the open part of the circles so that every student could see the projection if they turn away from their monitors. Owing to the position of the screen, taking notes or trying to use the keyboard when the instructor was doing the demonstration was difficult in the circular lab.

The Programming Environment The programming environment was more confined by the software site-license on campus than anything else. All machines in the labs had

Figure 9. The Layout of the Circular Lab.



Windows 3.11 and well-equipped with both Borland C++ and Turbo C compilers. The Turbo C environment was simple and easy to use. The Borland C++ environment was very similar and yet more sophisticated and complex. Since the programming environment is only one of the many tools in programming and the purpose of the lab assignments was to teach students how to program, the Turbo environment was chosen for its simplicity.

The Turbo environment or IDE (Integrated Development Environment) provided all standard features to help students in debugging which was done extensively in the labs. It provided a simple editor very similar to WordStar. Its debugger allowed users to watch variables change in a separate window, to set up break points, to examine the content of variables, and to step through or trace a program for sequential execution. In short, it had all the necessary features for program development. Thus, students were exposed to the debugger early so that they could use the debugging tools to help them understand the behavior of a program. However, this compiler did not have the program animator as in DYNALAB (Birch et al., 1995, pp. 29-33).

Teams In every lab, the number of students outnumbered machines almost two to one. Therefore, students are organized into teams. Each student had the choice of working alone, with one partner, or with 2 partners. To help students, facilitator, and lab assistants get acquainted, at least 15 minutes were used in the very first lab to build a climate of friendliness and informality (Newstrom & Scannel, 1980, pp. 39-41). Teams were formed by students themselves, and they stayed together for at least the first 8 weeks. Since most students were freshmen, most of them did not know other students in the first lab. Several new teams were formed with the consent of students after the first round of interviews in the ninth week so that the researcher could team students up based on age, gender, and prior experience.

Lab Assistants Lab assistants are very common in most lab situations. Each lab was assigned two seniors to assist the instructor, to answer questions, to fix hardware or network problems, to perform necessary tasks such as refilling the paper tray of printers. This arrangement helped the facilitator become the observer without running around the lab too much.

The lab assistants met with the facilitator before and after each lab. Before the lab, they learned about the content of the lab and tried to anticipate the different learning activities and potential problems. After the lab, they summarized the type of questions asked by the students. Questions related to their presence and impact were asked in the interviews.

Interview Questions

As in the tradition of naturalistic studies, the participants were purposefully selected for the interviews. In the age category, students of the following ages were chosen: 14 (just turned 14 when interviewed and the youngest in class), 16 (a sophomore in chemistry), 17, 18, 18, 35, 38, and 45 (the oldest in class). Other students were chosen for the interview because of their prior experience in programming or in C in particular. Some were chosen not from information in the assessment survey but rather observation data following the first several observations. Students who always finished labs first or last, students who showed frustrations in the lab, and students who were eager to help others were selected.

Each interview was done in the instructor's office. In the interview sessions, it was important to put students at ease. This was especially so for traditional age students

ease. This was especially so for traditional age students and students who were not talkative. In the first interview session, the instructor and the interviewee got acquainted by discussing things related to a few unguided questions (e.g. "Do you like computers?" and "Do you own a computer?"). Interviewee were also offered chocolate and soft drink to build a climate of friendliness and informality (Newstrom & Scannell, 1980, pp. 39-41). Most of the interviewees early talked about their own computer, about why they did not have one, or about how they were shopping for one and needed advice. After about 5 minutes, more structured conversations were organized. To correspond to the research questions (see p. 17), the following questions were used to direct discussions with the participants in the first round of interviews in the middle of the semester:

1. How does the transfer of concepts from lectures to labs take place?

   A. How long do you usually need to complete a given lab?
      How do you usually tackle a lab?

   B. How long do you usually spend on the write-up of the lab report?
      How do you get started?

   C. How did the labs help you complete the take-home programming projects?

2. What learning strategies do students use to learn syntax and semantics in the language C, and what are the perceptions of the result?

A. Did the lab help to clear up the usage of the following concepts? How?
   a. Nested lops
   b. Parameter passing: Both by reference and by value
   c. Function calls: void and otherwise
   d. Arrays
   e. Pointers and addresses

B. How did the lab activities help you remember or make some sense out of the syntax and semantics of the language C?

3. What are the students' attitudes toward computer science as a result of the lab experience?

A. Which features would you like to be added in the lab? Why?

B. How was our attitude towards programming influenced by the course of this class?

4. Why does the lab work or fail from the students' perspective?

A. Which features do you find helpful in a particular concept? Why? Please give examples of concepts you learned easily and those you had difficulty with.

B. Did anything in the lab hinder your learning? Why?

C. Which lab activities were confusing? Why?

The discussions of the interview sessions were taped recorded. These were later analyzed to uncover the common trends of participants' responses. A database was built for the purpose of understanding and analyzing the data collected.

Observations and both rounds of interviews coincided and were not separate phases. The first interview process started seven weeks into the semester. The process took three weeks. Observations began the first week and were continued until the end of the semester for a total of 15 weeks. The second round of interviews with participants was shorter and was conducted the week before finals week. In these interviews, they were also asked what one or two things they thought would improve the lab in any way. Throughout the course of the interview process questions were omitted and added because of the response from interviewees. For example, since "a shorter lab following each lecture" was a recurring themes from students, the last few students were asked what they thought of that idea even though they did not volunteer that idea. Since some students were asked to change their partnership situation after the eighth week (e.g. switch partners, work alone after having a partner, or work with a partner after working alone), different questions were

asked to different participants. Some of following questions were asked:

1. How do you feel about having a short lab after each lecture?
2. Did the lab assistants provide "better" help in the second half of the semester?
   Why did you ask less (or more) questions in the second half?
3. Why did you turn in incomplete lab reports?
   How do you feel about the write-ups?
4. Did the physical setup of the lab affect your lab experience?
5. Would you take another programming course with similar setup?

The interviews provided feedback and insights about learning how to program in the language C in the new computer labs from the students' who have just been through such learning experience. Combined with observations and questionnaires, an overall picture emerged of how students learn and what worked in the lab. The findings might help students and instructors in future labs. They may also provide a basic understanding of lab learning in a beginning programming course for future research.

CHAPTER 4

FINDINGS

Data were collected quantitatively and qualitatively from several sources. In the beginning of the Fall Semester 1996, questionnaires were administered in CS210 to gather information. Observations were conducted in all three lab sections of the class for 15 weeks. Interviewees were selected based on the initial questionnaires and the first seven weeks of observations. Mid-semester interviews started after the seventh week of the semester and were conducted for three weeks. At the end of the semester, another assessment survey was administered to all students while another round of short interviews were conducted.

Beginning of Semester Survey

A survey (see Appendix A) was given to all students in the CS 210 course at the beginning of class. Of the 155 students enrolled, 134 completed the survey. The questions on the survey were designed to find out students' age, gender, and prior computing and programming experience. While 54% of students had programming experience, only 9% had experience in C. On

the other hand, 50% had BASIC and 20% had Pascal before this course. Since Pascal and C have the common predecessor of ALGOL, these students who had Pascal or C experience were identified and observed to see if they behaved differently in the labs than those students without prior experience.

End-of-Semester Survey

At the end of the semester, students were asked to filled out the course evaluation. Only 134 out of the initial 155 students filled out the beginning of the semester survey. Several students dropped the course during the semester. The evaluation was completed by 132 students. On the evaluation, students were asked to rate the following 5 possible sources of their learning: lectures and text, programming assignments, tests, labs, and help from tutor(s) and/or instructor. They were asked to rank them from 1 (most important) to 5 (least important). The results of the survey are shown in Table 5.

The students expressed a clear difference in their rating of their preference for learning methods. The highest rating of the learning method was the newly

developed lab (2.36). Traditional lectures and text was rated second (2.51) with help from instructors/assistant third (2.72). While take-home assignments received a moderate rating (3.01), tests were not viewed as a useful learning method by students with an extremely negative rating of 4.39; three quarters (75%) rated tests as least important.

Table 5. End of Semester Survey

	Rating					
Method	1	2	3	4	5	average
Labs	44	36	17	30	5	2.36
Lectures and text	40	24	31	35	2	2.51
Help from tutors and/or instructor	20	42	35	24	11	2.73
Take-home Programming Assignments	23	25	28	39	17	3.01
Tests	5	5	21	4	97	4.39

Observations and Interview Results

Observations were conducted throughout the 15 weeks of the course. There were observable differences in students with respect to the physical setup of the lab,

the partnership patterns, the utilization of lab assistant, and the contents of the labs. The difference in time spent on each lab, the frustration or satisfaction of students with the lab, physical and programming activities were recorded. Students' perception on the same subjects were investigated in the interviews.

The Physical Environment

There were major differences physically between the two labs used by the three lab sections. The differences in size and layout translated into observable patterns in the demeanor of students. During the interviews, the physical setup also triggered more reactions by the students and some were extremely negative.

The Circular Lab. Two of the sections were scheduled in the "circular" lab. Students in the "circular" lab seemed comfortable and happy with the set up. They had enough room to work around their computer and to conduct discussions with their partners or their neighbors, who were students working on their own computers but in close proximity. The only problem with

this lab was that while the keyboards, monitors, and the mice were on the tables, the machines were mounted under the tables. Thus, students often hit the reset button with their knees inadvertently when they tried to sit down. This happened to machines under regular tables and under drafting tables including the facilitator's machine. Unfortunately, it took 10 to 15 minutes for each machine to come back up because a reset or reboot activated the network maintenance program which checked every directory on the machine and reloaded any file that was modified or deleted. "It's frustrating that we keep bumping into the reset button. We end up waiting forever for the machine to reboot and connect to the network."

Even in the larger "circular" lab, almost every student interviewed complained about the number of machines being too few and the number of students being too large. "More computers or less students would be nice." It was obvious from observation that even though 60'x 60' was a big room, 55 students were way too many for the space, and this large number of students presented a problem for learning in a hands-on situation. The problem was compounded by an unreliable

network. "Anything that requires the use of the printer and the network was terrible. One time I sent a file. It got printed 500 times, and no one could stop the printer. Others blamed me for messing up the network. It generated a major delay." The network went down periodically also presented a dilemma that was out of the facilitator's control. "We need a more reliable network so that printing a smaller file won't take half an hour, or the same file won't be printed infinite number of times. And, I had trouble downloading your programs from the net[work] several times." To combat the problem with the network, floppy disks were on stand-by so that at least students did not need to retype programs printed on the lab report. They got the programs from a disk instead of from the network. Nevertheless, when the network was down, using the instructor's floppy disk was not good enough because passing around the floppy disk with a large number of students took too long. When students could extract files from the server, it took only seconds. "I don't like the printing situation and the network, there were a couple times that we could not extract your examples in the lab and we had to wait for your disk." To

alleviate the printing problem, several machines were connected to dot-matrix printers. When the laser printers went down, students could take turns printing their files using those machines. However, the impact printing of dot-matrix printers created another problem -- noise.

Only one participant was completely happy with everything in the lab, "The lab is great as is. We are given plenty of time, space, and help. I have no complains."

In general, more computers or less student in the lab would improve the physical conditions of the lab. One student suggested one printer per machine. However, the feasibility of this depends largely on the fiscal budget. Physically, the "circular" lab could have handled one printer per machine because all machines were on individual tables.

The Rectangular Lab. One section of the lab was scheduled in the "rectangular" lab. Students in the "rectangular" lab seemed more frustrated with the physical set up of the lab. With only one printer, printing was a major problem. Table space presented

another problem because five machines were on one long table instead of each machine being on an independent table as in the "circular" lab. Students bumped into each other quite frequently.

Students in the "rectangular" lab complained even more about the physical set up of the lab. "More computers and more room would be nice so that we don't keep colliding into each other." Students mentioned the network problems as much as the circular lab. The complains about the printer were worse because there was only 1 printer for 26 machines and the lab had 50 students in a room which was about 25' x 55' compared to the circular lab with 2 printers, 55 students, and more than double the size (60' x 60'). Thus, the physical setup of the two labs played an important part in student learning and greatly influenced their frustration level. "The lab has too many students. It is too crowded. Even printing requires waiting for a long time. To make matters worse, students accidentally printed executable (non-printable) files a couple times, and no one could stop the printer or clear the print queue."

The tables were too close together for anyone to move around. Several students protested that it was difficult for the lab assistant to get to them because the lack of space between the big tables. One student summarized the situation nicely, "My partner and I try to avoid the machines in the middle of the row. Every time we get the middle machine, we try not to ask the assistant anything. Well, everyone in class prefer isle machines."

Overwhelmingly, the students argued on the need to 'make the lab smaller [less students], one person-one machine makes more sense to me." The number of students in a small room hindered students' learning in the lab in more ways than one. This crowdedness made all other problems seemed worse. "The lab is terrible, too many of us cramp in a small room, and the computers don't always work. The printing is slow and all mess-up when several of us send the same file or output to the printer. We don't know which is which." The sentiment of the students toward the "rectangular" lab was summed up by a student who was so frustrated than he cursed the computer repeatedly. He was more polite during the interview. "The class is too big, and the equipment

failure drives us crazy." When participants were asked in the interview about one improvement that they would suggest. Most in the circular lab said "more machines," and most in the rectangular lab suggested "less students."

The Programming Environment

No problems were observed with the Turbo C. The compiler was fast even for programs with several hundred lines. It was more than adequate in a beginning programming course even though it was not as sophisticated as DYNALAB (Birch, Boroni, Goosey, Patten, Poole, Pratt, & Ross, 1995, pp. 29-30). Unfortunately, the sophiscated compiler called education machine (E-machine) in the DYNALAB did not include the language C (p. 29). The debugger of Turbo C provided a good learning tool for the students. "The watch and step [commands] in Turbo C helped a lot. They give me the insights of how a program is run." Features in the debugger were covered early in the semester, and students were encouraged to make full use of them.

Students were also happy with the on-line help that they received from the compiler. One experienced

programmer reported, "I love the on-lone help, just <ctrl-F1> [press the control key and the function key 1 simultaneously], I get the syntax, semantics, and examples of a key word or a built-in function. I never even need to open the manual once." Another student who had taken this course once already last year compared the compiler on the VAX and Turbo C,

> The debugger was so difficult to use last year that I never used it, plus we didn't have lab so you couldn't show us how last year. Everything is making more sense this year. Maybe it's my second time, maybe it's the lab, or maybe it's the TC [Turbo C] debugger. I can run my program and watch all my variables change in another window simultaneously. It's been great. I think I'll pass [the course] this time.

A few problems did creep up. In the first couple weeks of the course, the compiler and the save environments, which save C programs and executable files automatically, were set up wrong. Without changing the linker to the correct directory, programs could not be compiled. Since instructors could not change the default set up on the lab, students had to make the change explicitly in the beginning of every lab. It was very confusing to first time users to make changes of which they had no idea. By the fourth week, the

personnel in the computing center finally put in the correct setup. However, by the time of the interviews, not even 1 student complained about the confusions of the first 3 weeks.

Learning Strategies

Many students commented on how they learned to program and some specifically mentioned how they made use of the labs. Their comments touched on a wide array of learning strategies they used in the newly developed lab concept.

Working in Teams. In the first half of the semester, teams of one, teams of two and, in a couple occasions, teams of three were formed. After the seventh week, some students were asked to reformat this team situation, that is students who have been working alone were asked to team up with a partner, students with a partner initially were asked to team up with another partner or to work alone. Even for those who did not get a chance to work alone in the second half, they were at least re-assigned with a different partner intentionally.

Students working with a partner tended to ask less questions of the lab assistant. One member of each team almost always did all the typing for the team, and this was always the member with more programming experience. Therefore, the teams were restructured half way through the semester to redistribute these learning elements.

When the students were asked about the partner situation, there were two opposing views about having a partner. Only three out of 21 said that they did not have a preference or felt comfortable either way. All three students finished their lab quite a bit faster than the rest of the class. All three tried both ways with and without a partner in the two halves of the semester.

The other students, on the other hand, felt strongly in favor of or opposed to the partnership situation. One engineering science student had the most positive experience,

> I like having a partner, especially my partner. We get along well, we are at about the same level in programming. We even got similar scores on the last two tests. We share a lot of programming ideas even outside the labs. Even though, in most case, we end up having different approaches in our assignments, we look at each other's programs and learn from each other.

> His partner recounted during her interview, I enjoyed having a partner that I got along so well with. We were so different that we learned from each other a lot. Since I got a partner, I asked the tutors less questions, and we helped each other on take home programs too.

Others just enjoyed the contact with other students. "I prefer to have a partner, someone I can discuss the problem with. Neighbors are okay too if you work alone." Another proponent of having a partner stated,

> Having a partner is wonderful because sometimes you sit there and say, 'What's wrong? What's wrong?' Working together is a great help because you can explain to someone what you know and have others explain to you what you don't know. When I worked alone during the early part of this semester, I discussed with my neighbors.

Less Experienced and More Experienced Team Pairings. Several students were purposely matched with more experienced partners. Some of them were glad that their partner helped them in relationships to points for the course. As with other studies (Ivey, 1992, pp. 109-111), there were observable benefits of matching less experienced with more experienced partners. The only observable benefit is the time required to complete the labs. Though students with a partner finished the labs in a shorter period of time than without a partner, just

"watching" or "listening" clearly were not enough. Almost all students paired up with experienced partners complained:

> I prefer not to have a partner, I don't feel I learn as much just by watching. Maybe I'm just a loner.
>
> My partner did things too fast. I had no idea what he did. After the first half [of the semester] I worked alone. I liked it a lot better because I had the keyboard, and I still had neighbors to discuss with and the tutors to help me.
>
> My partner goes too fast, and since he has more experience, he does all the typing. Sometimes he is so fast with the TC [Turbo C] commands that I have no idea what he is doing. Unless I can find someone with my kind of speed and experience, I prefer to work alone.
>
> I like individual labs better. I can discuss with my neighbors if I want. When I had a partner, I felt that I didn't learn as much though I received more points in those labs. Only one of us can be using the keyboard or the mouse at one time. Sometime my partner went so fast, I didn't know what he clicked.

The more experienced member of the team who were paired up with a less experience partner were also not entirely happy with the situation. Most of them concurred with the following statement: "My partner wasn't much help. I guess it may be a good thing if you have a partner who knows what he's doing." Another

student, who had a year of BASIC and was in engineering science, went even further by saying, "My partner really has no business in a programming class." Even though some of them did not feel as strongly about having a partner, they clearly prefer working alone in the lab because they had their neighbors if they craved discussions. Even a 33-year old student with much programming experience in other languages stated,

> With a partner, you don't get to try things you want. Watching it done is very different from doing it yourself because we all watch you do it on the big screen. It's vital that we try all the features provided by TC or features that may help at the instance we get stuck.

Another 24 year old computer science freshman echoed,

> I don't think having a partner is a good thing in this lab, maybe in another class. We may finish a project faster, but there are so many things that we are trying to learn. We are supposed to learn every part from the Turbo environment to C. I understand that in our field we need to learn to work with people, but we have a lot of technical things to learn before that.

Even students who enjoyed having a partner thought working alone in the lab played a major part in the

learning of how to program in a beginning programming class.

> I think you have to be able to do the labs on your own in order to understand programming. It's unfortunate that we have so many of us in the lab that some of us are required to have a partner whether we want one or not. I am glad that I tried both ways. I got along beautifully with her but I think we should work alone in the labs. Well, we work alone in home work assignments.

Similar Experience Team Pairing. The reactions toward having a partner depended largely on the gap between the programming experience of the partners. Students with similar prior programming experience worked well together. They also perceived their learning experience in the lab positively, and the programming language experience they had did not seem to matter. Four groups were formed in the second half of the semester based on their prior experience. They were observed carefully and were selected for interviews. One of the groups that always finished their lab the fastest and seemed the most enthusiastic in the lab were composed of two students with experience in two different languages -- one had 1 year in Pascal and the other had 1.5 years in BASIC.

> We just do the lab as in the lab report. [My partner, who has a background in Pascal] and I work well together. Most of the time, we don't even have any syntax errors when we compile. When I do the typing, he catches all potential problems before I hit <ctrl-F9> [compile command]. I do the same when he types.

It is important to note that the team members were willing to share the keyboard and the mouse when students were teamed with their peers with similar prior experience.

The other extreme group which almost always finished last was composed of two students with no prior programming experience, though one of them had some computing experience in word processing. One of them was an 18-year old computer science freshman and the other was a 38-year old sophomore majoring in business. As with the experienced group, they shared the burden of typing and using the Turbo C environment. Even though they needed more assistance from the tutors, they were not as frustrated as other groups that always turned in the lab close to the end of the allotted time. One of them summed up their approach, "We are just a couple of slowpokes, but we get the job done. We have gone

overtime only once. Thanks for your assistants; otherwise we would be lost."

There were a few students who hated both situations--with and without partner. They were selected as interviewees because they showed clear frustrations even in the first seven labs. Their typical response was that, "I don't think this course should be required in my major. I don't think I'll ever program again after this class."

Lab assistants. Lab assistants were vital to the students' learning in the lab. Each lab was assigned two computer science seniors to assist the instructor and to answer questions. The assistant to student ratio was about 1 to 27. Owing to the number of questions asked during the first 3 weeks, another lab assistant was added to each lab. Thus, the ratio was reduced to 1 lab assistant to about 18 students. The lab assignments were given to the lab assistants ahead of time so that they could get familiar with the lab as well as with the concepts students were supposed to learn for that particular week. The lab assistants in each lab were in great demand in all 15 weeks. They were almost always

with a student or a group of students. Not only did they answer questions, but they also provided feedback to the observation process, (e.g. what kind of questions were asked in each lab). Every participant in the interviews mentioned the importance of the lab assistants more than once.

Students' Point of View. Most students were extremely happy that lab assistants were provided. "I had very little idea what I was supposed to do in the first few labs if not for the tutors. They are wonderful especially [one particular lab assistant]." A student who took the class the year before and failed echoed this sentiment that , "having the tutors and you in the lab help a lot. It's far better than last year. I am doing much better this time. Things are not as confusing as before."

Some even went as far as declaring lab assistants as the best part of having a lab. A 28-year old business student commented, "The best part of having a lab is to have them there to provide instant help." Most of those interviewed agreed, "They are great, I don't think I can finish the labs as quickly without

them." Over 80% of the comments about lab assistants were very positive. Most of the students felt it would be good to have more of them around. "I would like to see more assistants because of the size of the labs. I need a lot of help. Sometimes it takes a long time before an assistant comes back to me. One tutor can't possibly help 20 some students."

On the other hand, several of the individuals and teams complained in the mid-semester interview that the lab assistants were "too" helpful. Since lab assistants have expertise but are not trained in the teaching process, there is a danger that they will give the learner the correct answer rather than teaching them how to get to the answer by solving the problem. The typical comments when assistants' help went overboard were:

> They are too helpful. Sometimes I ask a question and they finish the program for me. I felt embarrassed to ask again so I turned in a couple working programs without knowing why.
>
> Some are helpful. [However,] some tend to just get us unstuck without explaining what we were doing wrong.
>
> Every time I ask a question, [one of the lab assistant] took over my keyboard and made changes to my program beyond my comprehension. I usually avoid asking him anything.

There were also two complains about a few assistants who did not help students equally. One 18-year old computer science student complained, "You should have the tutors help students on an equal basis. If a tutor spends one-half hour with one student, then the rest of us feel neglected."

Because of these comments, a "no-touch" policy was instituted after the eighth week. Lab assistants were instructed not to touch students' keyboards or mice. They were asked to help students verbally or to illustrate ideas on a piece of paper. It took a few labs for the assistants to get used to the idea and to stick to that approach. During the end-of-semester interviews, not one single student mentioned that the assistants were "too" helpful. Thus, this approach alleviated the problem.

Lab Assistants' Point of View. Even though lab assistants were not interviewed the same way as students in the class, they met with the instructor before and immediately after each lab. They were asked about the difficulties and the most common questions in each lab.

They recognized the fact that oftentimes one member of each team tended to perform most of the typing.

At the end of the semester, lab assistants also commented that the lab feature of the course was a major part of students' learning in at least the language C and the programming environment. A 35-year old assistant compared the new lab situation with her own experience when she was a freshman in the introductory programming course, "I wish I had a lab when I was a freshman. Just learning how to compile and debug on the VAX almost prevented me from staying in CS [Computer Science]." Another senior was amazed by the progress of some students,

> I thought [one of the slowest students initially] was going to drop after the first couple labs. I can't believe the work he did toward the end. I don't know if I could finish the lottery program (see Appendix E) in an hour when I was a freshman. I think the weekly lab played a major part because they get the practice every week with help available instantly.

Assistant also kept track of the type of questions students asked. They noticed the number and the type of questions changed through the course of 15 weeks. "Towards the end, there were very few questions about the syntax of C or Turbo [the programming

environment]. Most of them have a running program but wrong output. I guess their logic is still weak." The type and frequency of the questions indicated the students had a good mastery of one programming skill but still the other.

Having the input from the lab assistants provided an angle that might have been difficult for the facilitator to observe.

Lab Manual. A complete set of Turbo C manual was available in the lab. The set consisted of three books: Getting Started, User's Guide, and Reference Guide. However, use of these manuals was almost nonexistence. During the 15 weeks of observation, only one of the three books was used twice.

Students were asked why they lacked interest in using the user's manual. The majority of the response indicated that they did not need the paper version of the manual because of the helpfulness of the on-line manual and the speed of it. One simple keystroke and they could examine the syntax and semantics of a situation or a keyword in C. One student explain,

> To find out more about the language C or the Turbo C environment, all I need to do is <ctrl-F1>. From the on-line help, I learned to set up four windows--one window for the program, one for the output, one for the debugger, and one for the error messages. I don't know why you even bother to have those books there.

The students were told that the user's guide had a lot of good examples. However, the typical student response was that "after I read the textbook to learn algorithm development and C, I can find what I am looking for a lot faster in the book." Another student explained, "I am so used to on-line manuals, those books are useless to me. When I brought my Visual C++, it came only with a CD but no books. Printed manuals may be out-of-dated." Thus, as a result of good on-line help, manuals in computer labs are quickly becoming obsolete.

The Time Factor. Students were encouraged to take their time and experiment with anything they desired after they finished their lab. Approximately 10% of the students always finished their labs within or a little more than an hour. About 15% of students always turned in their labs at the very end of the lab. The other 75% of students usually completed their labs within 1.5

hours to 2.5 hours. Although encouraged to use the extra time for additional learning, only about 20% of students would stay to experiment after turning in their lab assignments. One of the reasons for this was because they were advised not to stay in the lab to do their regular take-home programs. Since all lab assignments were due at the end of the 3 hours, lab assistants were told to concentrate on the lab task so that their efforts would not be diluted by anything else. Students were also advised that they should channel their effort toward a deeper understanding of the current concepts (Breuer & Zwas, 1993, p. 2). Even if students stayed in the lab to complete their take-home assignments, they were told that the lab assistants were there to help them with the lab first. Moreover, there was another lab, called Museum Cluster, which was set up for students to complete their regular home work assignments. The Museum Cluster was open from 7 a.m. to midnight every weekday, and one computer science tutor was assigned in the cluster area from 9 a.m. to 10 p.m. Thus, students who stayed after they finished their lab were strictly students who liked to experiment with the concepts of that particular lab.

There were clearly observable differences in teams and individuals who raced through the lab assignment compared to students who took their times to complete their assignment. Several students who always finished within an hour and left right after responded, "Our lab is from 3 to 6; our cafeteria opens only from 5:30 to 6:30, I can't afford to stay too long." A few other students who left within an hour stated that they had to go to work. They all had one common thread in their approach: They came into the labs prepared. One student who worked a night shift in a gas station explained his working situation:

> I get ready for each lab by following your reading assignments. The lab usually covers the concepts you go over that week so I have a good idea what the lab is about. I learned my lesson earlier. One week I got behind in my reading, I was late to get to work. I work from 6 to 2 [a.m.] so I have to leave way before 6.

Thus, for most of them it was out of necessary to get their lab done as quickly as they could.

However, the two youngest students who were 14 and 16 years old, considered the labs far better than the regular lectures.

> The pace of a traditional class is usually too slow. In the lab, when I feel comfortable with my achievement at any point, I can either leave or move on to something more exciting to me. For example, I started reading the graphics mode in Turbo C before you even started arrays [which is a more sophisticated programming concept covered toward the end of the semester). I like that a lot. I wish more classes were run like your lab.

There was no perfect time for setting up the 3-hour lab. With popular class hours being from 9 a.m. to 2 p.m. and with other classes that needed the instructional labs, the only time slots were 2 to 5, 3 to 6, and 4 to 7. Those were the slots that the labs were run.

Most students who were not in a hurry did finish the labs within 2.5 hours. Tension mounted greatly for some students during the last hour of the lab. If they were not close to being done, this group of students panicked. The most common strategy was the use of neighbors regardless of their teaming situation. Both students in one team recounted, "Well, if most other teams are done and we are not, we like to get a hold of a friend from another team and see if we are misinterpreting a problem. We either ask them or one of the tutors and have that problem clarified." Even

though students were told to progress at a pace compatible to their judgment and abilities, they did look around to see how other teams or individuals were proceeding. An 18-year old engineering science students, who earned the highest score in the class, added, "When half the class turns in the lab report and I am not close to done, I panic. It is like taking a test and struggling. Then you look around and realize that most people have left. That's not a good feeling." Thus, lab learning in a programming course added an unexpected peer pressure to some students because some of them unknowingly monitored the progress of other students in a negative fashion.

In the middle of the semester interviews, several participants expressed a similar idea with respect to when and how long the labs should be. Students attended their lectures on Monday and Wednesday followed by a 3-hour lab on Thursday afternoon. They speculated that the 3-hour labs were too long and that the labs covered too many concepts. A few of the participants suggested, "Instead of once a week for 3 hours, why can't we have a short lab right after each lecture. An hour and a half to 2 hours maybe." Another felt that

> a shorter and more focused lab would help me. After you cover 'if' statement in class on Monday, I like to get to the hands-on part in the lab right away so I don't have to wait until Thursday to try things out. Usually by Thursday [lab time], you would have introduced things like 'else if' and 'switch-case' before I can master simple 'if-else.'

The idea of "same-day lab" was introduced to other participants at the end of the semester interview. Overwhelmingly, 20 of 21 participants were extremely positive toward the idea. Most responses were:

> Definitely, if we can clear up a concept right after we see it in class, it'll save time for me to search for similar examples in the book. That's something I usually do.
>
> As long as the labs are not too long. Same day sounds like a good idea because we can learn a small chunk at a time.
>
> Excellent idea, except the labs can't be very long. It may be difficult to finish a whole program in 1.5 or 2 hours.
>
> I can see how it could help with the ideas fresh in our minds and apply them right away.

The only concern about that idea related to the timing between the lecture and the lab. "If it's right after the lecture, it'll work, otherwise it would be just like the current setup." Thus, the concept of same day lab may be an effective learning method if the labs can be scheduled correctly.

Background in Mathematics. The relationship between computer science and mathematics is as old as the field of computer science. Owing to the mathematics components in computer science, computer science instructors tend to believe that mathematics proficiency has a direct impact on learning how to program.

A few participants, mostly business students, considered their weak mathematics background as a reason for their poor performance, in a few of the labs in which they had to do simple analysis of algorithms in terms of the number of computational operations. They might have been intimidated by predicting the numbers which had a logarithm function in the formula (binary search). In another instance with nested loops, dependent variables and independent loop control variables caused major confusions among the same group of students. A check of prerequisites indicated that all students in the class had about the same amount of mathematics before this course. The prerequisite for the class at Montana Tech is college algebra. Thus, all students had a strong enough background in algebra for this course. Even though all majors at Tech require 2 years of calculus except for Business, most students in

the class were freshmen and were taking calculus simultaneously. Thus, all students in this course had similar prior experience in mathematics.

The business students were just intimidated in those two labs by their own attitude in mathematics. One 18-year old business student provided her explanation,

> I knew the number of guesses in the number game was related to log base 2. I just panicked when I couldn't remember the definition of logarithm. Luckily, you put that on the board after about half an hour into the lab. I could relate to the fact that each guess would eliminate half the numbers from the list. I just could analyze it mathematically in the lab. I just don't like math.

Contrary to some opinion, mathematics majors did not perform well in those two labs in terms of their grades, time of completion, and even their attitudes (Campbell & McCabe, 1984, pp. 1110-1112). One mathematics major offered his rationale, "I understand the lab and logarithm doesn't bother me. I just couldn't relate the guessing game and log base 2."

Programming is a required course which is taken by engineering science, chemistry, and computer science students in their freshman year. For business students,

it is a required courses to be taken in their sophomore year. Although the class had freshmen in most degree areas, almost all of the business students were in their junior or senior years. When participants were asked about their timing for taking this course, the participants blamed it on the subject matter and on the campus-wide conventional wisdom among business students that "we don't understand why business students need to take a programming course. We heard that this course is difficult and time-consuming so we waited. That's why you have us seniors in a freshman-level class." When they were asked if more mathematics would help, most of them responded, "Nothing will help. I don't like math. If I have to pick between programming and math, I'd rather program."

The attitude about math and programming might have been different among students with different majors. Nevertheless, their performance in the labs did not reflect their attitude. For example, some of the best lab reports were turned in by business students who hated both math and programming.

Cooperative Learning Environment. Some students used interaction among individuals or teams as a learning strategy. In the lab, students were encouraged to either work individually or cooperatively with a partner. It was the intent of the lab to avoid a lab structure that students would interact competitively. Nevertheless, several individual students and some teams paid much attention to the progress of other individuals and teams. Even though students were not graded on a norm-referenced basis to avoid "negative interdependence" (Johnson, Johnson, & Smith, 1991, p. 2), several of them perceived classes and labs as competitive venues. Two computer science students had almost identical comments though they competed with different students. "I like to be the first to complete each lab. Well, as long as I get done before [one other specific student], I guess I don't have to be first." When he was asked to comment on the significance of being first, he delineated,

> On a test, we get a numeric score so that I can measure my achievement against the rest of the class with the average and everything. In your lab, since most of us receive 10 out of 10 in every lab, the only way for me to chart my progress is my time of completion.

> The reason why I check on [this other student] all the time is because he's such a good programmer so if I get done before him, I am in good shape.

On the positive side, the majority of the students in the lab, however, interacted cooperatively with each other. Not only did they learn to work collaboratively with their respective partner or partners, but they also engaged in cooperative activities with other groups and individuals especially their neighbors or students in their immediate vicinity in the lab. Their relationship with their neighbors was very different from their partners. The neighbor system was neither promoted nor structured. It also lacked the basic elements of the structures in cooperative learning (pp. 5-7). Students perceived activities or discussions among neighbors as beneficial. As a matter of fact, several students asked during the first two labs if they could discuss the lab with students outside of their team. A 38-year old business student complained about the inadequate number of lab assistants but had positive comments about having "good" neighbors:

> There should be more tutors in the lab so that we don't need to wait 20 minutes for one of them to come back to my side of the lab.

> Sometimes I wonder if the network is down when my program doesn't get printed. Luckily, [my neighbor] always helps. One time she just printed her source code. Since her program got printed before mine, she figured that my machine had a bad connection. Sure enough, my machine was off the net. Other times, she'd help me debug when you and the tutors were busy.

Some students also stayed after they turned in their lab reports. This was not to do more experiments but rather to provide help to their neighbors. One student provided such good help to others that she became the only lab assistant in the following year. She was the only sophomore hired in the computer science department as a tutor or lab assistant.

Gender Differences. There were observable gender differences in terms of the students' willingness to ask for help in the lab. Male students were eager to ask the lab assistants whenever they had a question. Female students, on the other hand, felt more comfortable with partners and neighbors for consulting or assistance. They might be more reluctant to ask for technical assistance. One 35-year old business senior illustrated her reason,

> Lab assistants are good, but they tend to solve my problems too quickly for me. Most of the time, they solve the problems for me without explaining to me what I need to do. Then I get stuck again in the following step, so I like to try figuring things out by myself first. If I think about it long enough and still don't have the problem solved, then I ask.

In terms of the length or type of the questions, male students tended to ask short, direct questions. Female students had longer questions. When female students were interviewed and asked why they would go to their neighbor before lab assistants, one 18-year old computer science student responded, "Sometimes I have a problem explaining which part of the lab I don't get, and I don't want to take up too much of their time." A chemistry student concurred, "It takes me a while to explain to the tutor what I don't understand, so I usually think about it for a while before I ask."

Male students, on the other hand, raised their hands rapidly as soon as a problem was encountered. Thus, their questions tended to be short, and tutors could be seen moving from question to question quickly. If a lab assistant was in the vicinity of a group of male students, the assistant would usually answer

several questions asked by the same students intermixed with several questions by the students' neighbors.

Female students were somewhat more reflective and deliberate in terms of using other resources. For example, male students seemed to do more typing and compiling from observation. When they were asked why they re-compiled their program every time they made one simple modification, one male student responded, "I like to fix all my syntax errors so that I can run my program because programs won't run with syntax errors. After I get the program to run, I like to use the output to deal with any potential logic mistakes." After an error was found, a female student would be more inclined to make the necessary changes and look at other part of the program again before she re-compiled. Bernstein (1991) contributed the difference in behavior or comfort level of women in computing to their initial experience (p. 60). The past computing and programming experience of participants in this study agreed with Bernstein's study.

The Age Factor. Differences in learning strategies were observed among the students when they were

classified by age. The attitudes toward computing and programming were also different. The beginning of semester survey revealed that, younger adults , in general, had more computing and programming experience than older adults (i.e. students of the age 25 and older). Older adults' attitudes toward technology in general were not as positive as that of younger participants. One extreme view was from a 45-year old business major: "The computer is ruining our future. Not only is the technology controlled by a small group of elite like Gates [Bill], but computers are replacing people in many ways. I have been avoiding computers all my life." Not all older participants had this apocalyptic view of computers. Nevertheless, that attitude, unlike that found in the Morris' study (1992, pp. 72-75), was shared mostly by older participants and did not change much through the course of the 15 labs.

The time it took for nontraditional students to complete the lab assignments was noticeably longer than traditional students. The last few students to turn in the lab report in each lab were always non-traditional student. Nevertheless, there was no difference in the lab scores. One nontraditional student pointed out

that, "I am slow in understanding how all the pieces come together." Most non-traditional students responded, "I am also overwhelmed by all the things that I have to learn in the lab. Turbo C, the editor, getting files from the net." They also took more deliberate steps in completing the lab reports or experimenting with given programs. One participant described herself as "careful." They tended to read the programs before starting and worked out the possible outcome of the program. While traditional students tended to jump right in after the lab handout was given. The younger group compiled the programs more, made more modifications, and even had more printouts before they finally finished their labs. The younger group exhibited more experimentation in the lab.

The Language C. The use of the language C in the lab was a major concern. The conventional wisdom suggested that C was not the best language in an introductory computer science course (Dey & Mand, 1992, p. 11). Since there was no programming prerequisite for the first programming course at Montana Tech, to most participants, this course was their first programming

course. Only a few minor problems related to the language C occurred in the lab. There were no major complains about the choice of language from students. Two students suggested the language C should be dropped so that the language Java could be taught as their first language.

As to the syntax of the language C, the use of the semicolon in C presented a slight problem to students who had BASIC or FORTRAN in high school. Semicolons are used as statement terminators in the language C. Understandably, students had Pascal experience who did not seem to be bothered by the use of semicolons because semicolons are also used as statement separators in Pascal. A few students who had only programming experience in BASIC complained that the rule of semicolons was confusing. They were accustomed to having the line feed or return to separate statements. As a result, they either used too many or too few semicolons. The compiler only picked up the problem when too few were used. When that happened, all the students had to do was to add semicolons wherever the compiler suggested. On the other hand, because of the flexibility of C, too many semicolons would not trigger

any message from the compiler. Unfortunately, the extra semicolons sometimes changed the meaning the those statements. This took students much longer to debug. On some occasions, even lab assistants overlooked the extra semicolons. In the process of mastering the use of semicolons, a few participants got frustrated; this was expressed as "I don't like C. BASIC would never have given me troubles like that."

Most students had trouble with passing parameters especially passing parameters by reference. It was because of the confusing nature of C in the use of symbols ampersand (&) and asterisk (*). An extra lab was designed to help students with problems unique to the language C.

Overall, the language C did not present any major problem in the lab. Thus, C did not have any significant negative impact toward students' learning how to program as has been suggested in some of the literature.

Write-ups. Each lab assignment consisted of several parts that were designed to help students reflect on the concepts covered in the lecture. The

write-up part of the lab was intended to take up no more than 25% of each lab out of 3 hours. Nevertheless, actual time spent on each write-up ranged from 10 minutes to an hour. "Does the result correspond to the 'ham and cheese' example in class? If yes, in what way?" (see Appendix C) is a typical question in the write-ups. A logical "and" operation in programming was not more complex than ordering a simple sandwich at the deli. All students had to do was to relate a program to a concept. There were two to three write-up questions per lab assignment.

To some eager programmers who only wanted to do the coding part, the last thing they wanted to do was documentation. "Sometimes I don't understand what you ask and don't know what to write. If I can finish the programs in an hour, I don't think I need to explain anything. At least the hard copy of the program or my test runs should work," said a 17-year old computer science freshman who just wanted to practice the implementation part of programming in the lab. The write-up part was designed to help students learn the why instead of just the how regarding programming concepts.

On the other hand, 20 out of 21 participants responded positively to the write-ups. They considered write-ups were a vital part of learning how to program. However, they took different approaches dealing with the write-ups. One quarter of the participants would finish the whole lab assignment before they started doing the write-up part of the lab report. A chemistry student explained,

> I like to have all the programs or design done before I answer those questions. I like to think about what I have learned in that lab so that I can see the big picture. Your lab write-ups are far shorter than my chemistry ones. I usually have to spend the night to complete chem lab reports.

Other students echoed this view and considered the write-up part as the last steps of the labs. "Sometimes I don't quite get what you are asking until I finish all the steps in the lab. Even step 2 relates to step 1 directly, I wait till the end."

The majority of students and participants did the write-up parts of the lab as they progressed through the labs. They simply followed the steps layout in the lab. Thus, when they were asked to answer a question following a program, they just did so without thinking much about it. "The step by step instructions and

write-ups are there, so I just tackle them one by one." Others took a more deliberate approach,

> The write-ups are usually there to break up programs or experiments so that I can think about what I am learning before I move to another activity. I use the write-up as an indicator. If I can't do the write-up in Step 4, I don't start Step 5.

The observations in the lab, interviews with participants, and grading the lab report all revealed that the write-up part of the lab assignment was a crucial part of learning how to program. The write-up were also excellent feedback to the instructor on how well the class or individual students were doing. However, owing to the time constraint, some students rushed through the lab and turned in sloppy lab reports.

In the interviews, several students expressed a preference of post-lab write-ups instead of in-lab. They wanted to turn in the lab report the following day instead of at the end of each lab. "If I had more time to reflect on what I am learning, I would do a lot better on the write-ups." Other students compared the new lab in programming with labs in other disciplines. "My chem lab reports are due the day after so that we have time to think and write. Doing everything in 3

hours seems a little bit rush." When participants were asked if take-home lab report would affect their take-home assignments, most of them suggested the number of assignments should be reduced.

Teaching and Learning Activities

Most programming teachers believe that laboratories are effective because well-designed experiments in labs offer a mode of learning that complements classroom teaching (Hartel & Hertzberger, 1995, p. 13). Thus, courses supported by short and relevant assignments are more effective than courses without such laboratories (p. 17). It is, however, up to the instructor to devise appropriate activities to facilitate the best learning for the situation.

The lab activities were designed to enhance learning programming in C in the newly designed labs at Montana Tech. The activities could be divided into three major categories of Record and Explain, Experiment and Discover, and Design and Justify. Students shed light on the impacts of each activities with candid examples.

Record and Explain (R&E) Simple programs were given as part of the activities in almost every lab. The activities were designed to stimulate the sensitivities of students' ability toward subtle differences in simple programs in C syntax and semantics. The observations indicated that the students liked R&E. "By running your programs, I learned at my own pace. I usually just use the output to understand the behavior of your program. I felt I learned C pretty well." The purpose of R&E could be best summed up by an engineering student:

> By fixing errors in your program, I learn the syntax and semantics of C without memorizing all the rules in the book. The compiler is excellent with syntax mistakes. The semantics problems take a little longer, but the output of the program is usually good enough to reveal them.

Several participants mentioned one particular lab in which the sum of integers from 1 to 10 was supposed to be in the variable sum as in figure 10.

Figure 10. Program Segment that Produced the Wrong Sum

```
sum = 0;
for (counter = 1; counter <= 10; counter++);
     sum = sum + counter;
```

Even student with previous programming experience did not immediately see the extra semicolon, which is shaded in Figure 10, after the right parenthesis. When the variable sum was printed after the for-loop, they found the sum was 11 instead of 55, the correct answer. They were asked to explain the phenomenon in the lab report. By using the debugger, they realized that the for statement was executed 11 times as expected but the statement, "sum = sum + counter;", was executed once instead of 10 times. It was after the variable counter had been incremented to 11 that the sum statement was executed. By explaining the phenomenon, they understood that the extra semicolon changed the meaning of the loop and caused the loop to execute 11 times without the body of the loop which was supposed to add the value of counter to sum 10 times. They had to explain that for the first 10 times of execution in the for loop, the statement, "sum = sum + counter;", was not involved as the program had intended. After their explanation, they were asked to fix the problem as documented in figure 11.

Students liked R&E activities because they provided a natural break in terms of pacing for the students in the lab. One students who enjoyed R&E activities reflected, "Even though I like all the activities in

Figure 11. Program Segment that Produced the Right Sum

```
sum = 0;
for (counter = 1; counter <= 10; counter++)
     sum = sum + counter;
```

your lab, I like the Record and Explain best because I am forced to stop and think what why the programs behave in a certain way." Thus, R&E activities were successful in helping students with the syntax and semantics of keywords as well as simple programming concepts in the language C.

Experiment and Discover (E&D)

Experiment and Discover activities were designed to encourage students to modify the program or programs in the lab report. All the programs were available to them on the network. Thus, students could download needed programs to their computers without retyping any of the

programs. Students were asked to modify the programs to perform a specific task or just to run the programs in various ways. A chemistry student compared the E&D activities in CS 1 lab with her chemistry labs: "Experiment and discover activities are my favorite and that's the spirit of experiment in a laboratory." Another student concurred, "Like I said a couple of months ago, I like modifying your programs [E&D activities] the best, but I feel good even with doing the whole program in the lab now."

Other students liked the part of E&D that required them to run the programs in various ways and discover a concept or an algorithm. "I like experiment and discover activities, especially games like guessing game that's similar to the Price is Right [the TV show] to demonstrate binary search. Who says watching TV is bad!" Students comprehended the algorithm faster by learning activities that were not as dry as lecturing (Bienat, 1993, p. 11).

Other participants liked E&D because they could use the programs in the lab as examples.

> To experiment with your program is my favorite because we have all the necessary programs there and knowing that even if we mess up we

> can always go back to the beginning of the lab. And I learned a lot about programming style and logic from those examples.

Design and Justify (D&J)

Design and Justify (R&J) activities, in a way, were the most difficult. Some students "feared" them because of they involve the creative side of programming. D&J was not language specific. Students were asked to design an algorithm to complete a simple task, for example, print all possible tickets in a lottery (see Appendix E).

In terms of time of completion, D&J generated the biggest difference. When D&J was the main activity in the lab. The time required to complete the lab for each student varied a great deal. A small group of students of three to five, turned in their lab reports and their programs in half an hour. On the other hand, 15 out 50 students still struggled at the end of 3 hours. Since the creative process was essential in programming, D&J provided the instructor a chance to guide students toward a solution and in some cases the most efficient one. For example, student were asked to print out all possible outcomes of drawing 3 out of 10 balls labeled

from 'a' to 'j' as described in lab assignment 10 (see Appendix E). Most students came up with a working solution in a matter of minutes. Some started experimenting with algorithm that they designed by doing a hand trace, some converted the algorithm into a C program, and some decided to make their solution more efficient. Since some of them had their solution done in half an hour, they were asked by the lab assistants or the instructor to justify their solution in their lab reports. For those students who had a solution but not the most efficient, they were given a hint about the number of outcomes they generated. There should be exactly 120 tickets. Program segment in Figure 12 was the most common among those who tried to turn in their report within the first hour.

Figure 12. Algorithm 1 for the Lottery Program

```
1. generate letter form 'a' to 'j' called ball1
2. generate letter from 'a' to 'j' called ball2
3. generate letter from 'a' to 'j' called ball3
4. if ball1 <> ball2 <> ball3 then print
                    ball1 ball2 ball3
<> means not equal
```

Though the algorithm worked, it generated 1000 (10x10x10) outcomes. Since only 120 tickets should be printed, they were asked to eliminate unnecessary configurations by generating only the configurations needed. Most students came up with the algorithm in Figure 13 on their own.

Figure 13. Algorithm 2 for the Lottery Program

```
1. generate letter form 'a' to 'j' called ball1
2. generate letter from ball1 + 1 to 'j' called
   ball2
3. generate letter from ball2 + 1 to 'j' called
   ball3
4. print ball1 ball2 ball3
```

Several students needed the lab assistants' demonstration in order to get the most efficient solution. Lab assistants were told to show those students that their solution in algorithm 1 generated configurations: 'aaa', 'aab', 'aac', …, all the way to 'jjj.' Clearly 'aaa' was necessary because once ball 'a' was drawn, there were only 9 balls (from 'b' to 'j') left. Almost all students changed their algorithm after the illustration was made. The purpose of this lab was

to learn nested loops as well as loop control variables that were dependent.

Several students had a problem with D&J activities because of the time limit. Some felt that "Design and justify activities usually take too much time, I learn just as much with the other two kinds [of activity]." Another business students gave almost an identical comment: "The designing of algorithm takes too much time. It's hard to complete a program in the lab. I prefer other lab activities that I know I can get done."

On the other hand, some computer science students had a diametrically different view. "I like designing algorithm and writing whole programs. I like the creative part of programming even though your examples are good. I guess that's why I pick CS [Computer Science]." Some also enjoyed the challenge: "I like to start from the beginning of the whole program, it's more challenging."

For slower students, time presented a major problem with this activity. Nevertheless, with the availability of help, participants and other students did not complain as strongly as they about the physical setup of the labs.

Interview Summary

The major finding from interviews and end-of semester survey was the fact that students considered the labs to be the most significant learning element of the course. Both qualitative and quantitative data from the study supported that.

One of the major findings from both the mid-semester and end-of-semester interviews was the importance of lab assistance. Both positive and negative comments regarding lab assistance shed light on how assistance should be offered.

Another important finding was the way that work groups were formed. Participants had strong opinions regarding what kind of partner they should have. Clearly, if physical constrains would not permit them to work alone, they definitely prefer to have a partner with similar prior skills. Their learning experiences were affected by that directly. Besides "official" partners, neighbors played an important role in their learning process for participants in teams or working alone.

Strong feelings were also reported in relation to the physical settings of the lab. On the other hand,

Turbo C programming environment did not present much problems for students. Age, gender, and background in Mathematics affected their learning strategies only slightly. Printed lab manual did not have much impact at all.

The students expressed ideas in the interviews concerning the structure of the lab. They discussed a preference for same-day lab and take-home lab reports. Participants clearly preferred the idea of mastering one concept at a time by having a short lab right after a lecture. They also favored having more time to finish the write-up part of the lab report.

CHAPTER 5

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

Summary

The computer science department at Montana Tech has designed and implemented a laboratory component in the freshman programming course. This naturalistic case study was designed to investigate how students learned to program using the language C in the lab environment. The study employed both qualitative and quantitative methods. Two assessment surveys were done. Students were observed in the lab for 15 weeks. Two rounds of interviews were conducted during the semester. Research questions related to student learning strategies, the physical environment, the programming environment, and teaching and learning activities.

During the first week of the semester, a student profile assessment (see Appendix A) was administered to students in the course. The results were analyzed and used to select participants for in-depth interviews.

Observations were started from the first full-week of the semester in the lab and continued for all 15 weeks of

the lab. Approximately 150 students were observed in the lab setting.

During the first 7 weeks of observation, students were selected purposefully for the interviews. These 21 students participated in two in-depth interviews. Interviews started in the middle of the semester. The same participants were interviewed again at the end of the semester to affirm several emerging ideas. Another student assessment survey was done during the last week of instructions right before finals week.

One focal point of this study was to reveal students' learning strategies in the lab throughout the course of the semester. Several learning strategies were identified. Many students relied heavily on lab assistants to aid them in their learning. Thus, the availability and quality of assistance were crucial, and the delivery of assistance was modified during the course of this study based upon the observation and interview data.

All students enjoyed having neighbors for discussions, consultations, or simply moral support. Thus, the closed lab concept was a clear success. The closed lab provided a cooperative interactive learning environment though it lacked the structured elements in classic cooperative

learning (Johnson, Johnson, & Smith, 1991, pp. 5-8). More importantly, the formation of teams in the lab had major effects on students' learning. First of all, with limited space and machines and the large number of students, teams consisting two or three students were formed out of necessity. If the scheduling of the lab permitted, most students preferred to work alone to ensure their understanding of every concepts at their own pace. If students had to be teamed, most participants favored a partner with similar prior experience. The composition of teams affected the learning of programming.

The size and even the physical layout of the lab prompted numerous comments from respondents. The difference between the two rooms used as labs, in fact, related to the number of complains that were heard from the interview participants but all students in the labs.

Another focal point was the examination of the three teaching and learning activities: Record and Explain (R&E), Experiment and Discover (E&D), and Design and Justify (D&J). R&E and E&D activities received overwhelming positive comments. Both activities seemed to help students learn the language element of programming. Comments on D&J activities, nevertheless, received mixed blessings. Some

participants thought D&J activities related very well to the problem-solving or algorithm side of programming. Some students complained about the time it took for the activities. Because of the time constraint, D&J activities tend to elevate levels of anxiety among some students especially in a few labs when the D&J activities were too long. The culprit of the anxiety could have stemmed from the instructor's inability to properly estimate the length of the exercise problem and the unpredictability of the computers and the network.

Conclusions and Recommendations

The laboratory model for the freshman programming course has been developed and tested. The newly developed lab was well-received and provided an excellent element which augmented the traditional instructional elements such as lectures, examinations, and written and programming assignments. Students have not only embraced the lab concept as a teaching and learning tool in programming, but they also considered the new element more important than all the traditional instructional elements.

The success, however, has not been without cost. Owing to the lack of graduate teaching assistants, the added

contact hours in the lab have been the source of debate within the computer science department regarding teaching load. The 3 lab sections occupied 2 laboratory facilities for a total of 9 hours a week. If the policy of one student per machine were to be implemented, there would need to be 6 lab sections and thus 18 lab hours a week in the same 2 lab facilities. With two lab assistants per lab, the computer science department needs to budget additional part-time money (about $3500 each semester @ $6.5/hour) to pay the lab assistants.

Elements within the lab such as learning strategies and learning activities play different roles in the newly developed lab. It is the responsibility of the course instructor or lab facilitator to ensure the lab conditions are favorable for learning.

Beginning-of-Semester Assessment

Students from several majors took the introductory programming course because it was required. Previous experience in programming or even computing cannot be assumed for a freshman course. It is especially true for non-computer science majors and non-traditional students. Thus, a beginning programming course must be designed to

anticipate a heterogeneous audience, and no assumption of prior programming experience should be made.

End-of-Semester Assessment

From the end-of-semester assessment questionnaires, students considered the lab component of the course extremely valuable. In fact, the lab part of the freshman programming course was rated more important than any other components. The end-of-semester interviews with participants confirmed this finding. The newly developed lab was a success. Thus, the lab component should be incorporated in a beginning programming course. There may still be a few bugs that needed to be work out, and perhaps, new components can be added to enhance learning programming in the lab.

Partners and Neighbors

The two rooms in which the labs were conducted could not handle the number of students physically. The lab environment created a collaborative learning climate for the exchange of ideas (Knowles, 1984, p. 15). The sheer number of 50 students with only 2 lab assistants unintentionally encouraged students to cooperate and collaborate.

On the other hand, participants had numerous opinions about how the collaboration should be done. Most of them preferred working alone than with someone with dissimilar prior programming experience. The teaming method that students paired with whomever they wanted was a bad idea. When students outnumbered computers, more deliberate method should be used for choosing or assigning partner to better the collaborative learning climate. Students cared more about the prior programming experience of their partners than other factors such as gender, age, or major. Cases should be avoided where one partner "did not know what was going on" because the partner "controlled the keyboard and went too fast" or where one partner feels the other partner "had no business in a programming course."

Students should work alone in a beginning programming course if it is possible because "watching it done is different from doing it" and students need to "try every tool in the Turbo environment to every concept in C." For heavy task-oriented activities, groups may side-track energy toward relationship tasks instead of toward the task. Thus, labs need to have enough machines for each student to work alone.

Physical Environment

Regardless of the partnership circumstances, the physical environment can play an important role in learning how to program in the lab. The physical setup of the labs including the computers must be viewed as resources, and the facilitator must encourage students to devise strategies to utilize them (Brookfield, 1986, p. 102). In this study, participants made numerous comments about the physical environments. Physical comfort is clearly important if students are captive for 3 straight hours. Enough space must also be provided to encourage discussions among teammates or neighbors. Even the position of each computer or at least the reset button can play a role in learning how to program.

On-line Manual Versus Printed Manual

With the storage technology today, students will clearly choose on-line manual over printed ones because the two forms of manuals are identical in details now. The difference between the two forms is the time it take to find a command or the syntax of a keyword. On-line manuals are more efficient than the printed ones for students in a programming class. The on-line manual provides one key

stroke to get to the page where the printed manual would be. More importantly, all programming examples can be copied and pasted onto the programmer's working window. As a matter of fact, students in the lab used the printed manuals only twice in all 15 weeks.

Post-lab Write-ups

The write-up portion of the lab report is designed to help students to get in-depth insights about an algorithm, a whole program, or a program segment. It is a major learning part of every lab assignment. For some students, the write-ups are used for reflective learning, and thus should not be done in a hurry. Some students preferred to turn in the lab report together with the write-ups the day after the lab. More in-depth questions could be asked in the lab report if a post-lab write-up is used in conjunction with an in-lab one. The in-lab and post-lab write-ups are not mutually exclusive. Their usage can be based on the lab particular activities. Thus, to alleviate unnecessary anxiety for students and to enhance reflective learning, write-ups should be done as post-lab assignments.

Teaching and Learning Activities

The three activities of Record and Explain (R&E), Experiment and Discover (E&D), and Design and Justify (D&J) can be used individually or in any combination depending on the material to be learned. R&E and E&D received overwhelming positive comments. Comments on D&J, on the other hand, were mixed. One cognitive advantage of learning in the lab is the way it forces students to experiment in a structured fashion. For participants who were initially reluctant to modify the given programs, these type of activities encouraged them to find the joy of eventually understanding other people's code.

Students may feel overwhelmed when they are given a whole program to complete in 3 hours as in the D&J activities. If they can complete their design and go over their ideas with one of the lab assistants or facilitator to make sure that they are on the right track, turning in their program or their algorithm with justification on the following day may facilitate learning. This approach to doing D&J activities coincides with the post-lab write-ups idea.

Lab Assistants

The Role of Lab Assistants. Lab Assistants are important to the success of the lab in which students learn how to program. There were numerous comments on the issue of assistance in the lab. The feedback from participants were mostly positive. Some of the negative comments were dealt with immediately such as the "no-touching policy," in which assistants were instructed not to touch students' keyboards or mice.

Weekly Meetings. Weekly meetings play a vital role in the success of the lab program. Lab assistants need to be familiar with each lab before it is given in order to assist students efficiently. The weekly meeting can also be used to adjust the method of assistance. Some positive or negative comments can be dealt with instantaneously. Since these assistants are not trained teacher, improvements should be made from week to week. For example, the "no-touching" policy was implemented after the eighth week of the new lab because some assistants took over the keyboard and finished the programs for the students without explaining to the students what caused their programs not to work.

In the beginning of the course, lab assistants should be trained to provide assistance equally. They should be advised not to take over students' project and especially not to control their keyboard. They must also not change the students approach drastically to solving the problem unless the approach is total wrong. The lab assistants must understand individual student's solution to the lab problems and guide the student to complete each lab problem. They should not introduce concepts that have not been covered in the lectures to improve the program on which the students are working.

Same Day Labs

In the mid-semester interviews, almost half of the participants mentioned that shorter labs with less concepts covered would alleviate some of their anxiety toward the amount of work in each lab. When the idea was introduced to other participants, they all agreed with the idea and said they were eager to try if it could be implemented in the following programming course. The format of a programming course in which two separate lectures on 2 different days followed by a 3-hour lab on yet another day may not be the best for learning. Instead of a full 3-hour lab each week,

2 shorter labs may be used every week, and each lecture is followed by the short lab which covers only the concept of the day. Thus, hands-on activities can be short and students can learn in small steps. The lectures and labs can be more coherent. Both students and instructors can receive daily feedback. With this approach, the lab portion of the course can become the focal point rather than the supplement to the lectures.

Recommendations for Further Research

Closed-laboratory in the introductory programming course has been proven to be an effective learning tool (Thweat, 1994, p. 81). This study confirms that it also works in a small 4-year engineering school. This study also describes how students learn in this situation. Future research could examine if this lab model could be applied to other computer science courses especially lower division programming courses. Additional research would be beneficial related to the effect of learning style has on students in a programming lab (Marshall, 1995). More could be learned about students' learning strategies with respect to learning activities and whether explicit instructions in

learning strategies would be effective. It may also be worth-while to do a study on same-day labs.

Internal elements in the lab are, however, not used universally. Computer science lab infrastructure advocated by pioneers in the field should also be investigated. For example, the DYNALAB (Birch & Associates, 1995) provides visualization of when programming statements are executed, and "course-ware" (Lin & Associates, 1996) provides interactive programs for students to experiment with different concepts and visualize the walk-through of algorithms. One could also examine ideas such as using a subset of a language instead of the full implementation, as a teaching tool as in Education C (Ruckert & Halpern, 1993, pp. 6-9).

The Future

The laboratory component of the beginning programming course at Montana Tech represents the future direction of the computer science department. The hands-on pragmatic approach is consistent with the mission of Montana Tech as well as with its other engineering curricula. By learning more about how students approach the new lab, instructors can select strategies and design activities which will

improve learning. By gaining more insights about lab learning in the field of computing, similar approaches can be tested and applied to other courses.

Professors who went through computer science programs when there was no lab may not see the need for the new paradigm. Other instructors who use lectures as the only form of teaching may also be uncomfortable with the change. More importantly, the infrastructure is still being developed and most ideas have not been tested as in other science fields with a longer history than computing. Lab books for computer labs are scarce. The vast number of different languages being used may have been one of the reasons why publishers are slow and reluctant to lend a supporting hand.

There is a movement in the field of computer science to utilize labs to help students learn. The job of the educators who have used the lab approach to enrich students' learning experience is to inspire colleagues through meetings within their departments, computer science conferences, and the literature; to train lab assistants; to test other appropriate lab learning activities; to convince administrators to support the approach financially.

REFERENCES CITED

Abrial, J. R. (1980). The specification language Z: basic library, programming group. Oxford University, United Kingdom.

ACM Curriculum Committee on Computer Science (1979). Curriculum 78: Recommendations for the undergraduate program in computer science. Communications of the ACM, 22(3), 147-166.

ACM Education Board (1989). Computer science as a Discipline. Communications of ACM, 32(1), 9-23.

ACM/IEEE-CS Joint Curriculum Task Force Report (1991): Computing Curricula 1991. Communications of ACM, 34(6), 69-84.

Adams, F. (1975). Unearthing seeds of fire: The idea of Highlander. Winston-Salem, N.C.: John F. Blair Publisher.

Anderson, C. L. (1991). Educating beyond the campus. Human Ecology, Winter Forum, 16-19.

Anderson, J. R. (1980). Cognitive psychology and its implications. San Francisco: W.H. Freeman and Company.

Backus, J. (1976). Programming in America in the 1950s--Some personal impression. International Research Conference on the History of Computing (pp. 125-136). Los Alamos Scientific Laboratories, New Mexico.

Bernstein, D. D. (1991). Comfort and experience with computing: Are they the same for women and men?. Special Interest Group Computer Science Education (SIGCSE) Bulletin, 23(3), 57-64.

Bierna, M. J. (1993). Teaching Tools for Data Structures and algorithms. Special Interest Group Computer Science Education (SIGCSE) Bulletin, 25(4), 9-12.

Birch, M. R., Boroni, C. M., Goosey, F. W., Patton, S. D., Poole, D. K., Pratt, C. M., & Ross, R. J. (1995). DYNALAB -- A Dynamic Computer Science Laboratory Infrastructure Featuring Program Animation. Twenty-sixth SIGCSE Technical Symposium on Computer Science Education (pp. 29-33). Nashville, Tennessee.

Bloom, B. S., & Associates. (1956). Taxonomy of educational objectives handbook: Cognitive Domain. New York: Mckay.

Booch, G. (1994). Object-oriented analysis and design with applications. Redwood City, CA: The Benjamin/Cummings Publishing, inc.

Breuer S., & Zwas G. (1993). Numerical mathematics: A laboratory approach. Cambridge University Press.

Brookfield, S.D. (1986). Understanding and facilitating adult learning. San Fransico: Jossey-Bass.

Brown, A. L., & Palincsar, A. S. (1989). Guided cooperative learning and individual knowledge acquisition. In L. B. Resnick (Ed.), Knowing, learning, and instruction. Hillsdale, N.J.: Lawrence Erlbaum Associates, Inc.

Bryant, R., & Palma, P. D. (1993). A first course in computer science for small four year CS program, A Quarterly Publication of the Association for Computing Machinery Special Interest Group on Computer Science Education (SIGCSE Bulletin), 25(2), 31-34.

Bruce, K. (1991) Creating a new model curriculum: A rationale for Computing Curricula 1990. Education and Computing, 7, 23-42.

Campbell, P. F., & McCabe, G. P. (1984). Predicting the success of freshmen in a computer science major. Communications of ACM, 27(11), 1108-1113.

Conti, G. J. (1977). Rebels with a cause: Myles Horton and Paulo Freire. Community College Review, 5(1), 36-43.

Conti, G. J., & Fellenz, R. A. (1991). Teaching adults. Tribal College, 18-23.

Cross, K.P. (1981). Adult as learners. San Fransico: Jossey-Bass.

Darkenwald, G. G., & Merriam, S. (1982), Adult education: Foundations of practice. New York: Harper & Row Publishers.

Denning, P.J. (1992). Educating a new engineer. Communications of the ACM, 35(12), 82-97.

Devore, J., & Peck, R. (1996). Statistics: The exploration and analysis of data. St. Paul, MN: West Publishing Company.

Dewey, John (1938), Experience and education. New York: Collier Books.

Dey, S., & Mand, L. R.(1992). Current trends in computer curriculum: A survey of four-year program. Special Interest Group Computer Science Education (SIGCSE) Conference Proceeding (pp. 9-14), Kansas City, Missouri.

Dijkstra, E. W. (1980). A programmer's early memories. In N. Metropolis, J. Howlett, & G. Rota (Eds.), A history of computing in the twentieth century. New York: Academic Press.

Elias, J. L., & Merriam, S. (1980), Philosophical foundations of adult education, Krieger Publishing Company.

Foster, L.S. (1992). C by Discovery (2nd Ed.), El Granda: Scott/Jones Inc., Publisher.

Freire, P (1973). By learning they can teach. Convergence, 4(1), 1-3.

Friedman, L. W. (1991). Comparative programming languages. Englewood Cliff, NJ: Prentice Hall.

Gagne, R. W. (1966). The conditons of learning. New York: Holt Rinehart Winston, Inc.

Gay, L. R. (1992). Educational research: Competencies for analysis and applications. New York: Macmillan Publishing Company.

Geitz, R. (1994), Concepts in the classroom, programming in the lab. Twenty-fifth SIGCSE(Special Interest Group on Computer Science Education) technical symposium on computer science education (pp. 164-166). Phoenix, Arizona.

Gersting, J. L., & Gemignani, M.C. (1988). The computer: history, uses & limitations. New York: Ardsley House.

Gibbs, N.E. (1989). The SEI education program: The challenge of teaching future software engineers. Communications of ACM, 32(5), 594-605.

Gleick, J. (1988). Chaos: Making a new science. New York: Penguin Books.

Guba, E. G. (1978). Toward a methodology of naturalistic inquiry in educational evaluation. Los Angeles: Center for the Study of Evaluation, UCLA Graduate School of Evaluation.

Harrisberger, L., Heydinger, R., Seeley, J., & Talburtt, M. (1976). Experiential learning in engineering education. Washington, D.C.: American Society for Engineering Education.

Hartel, P. H. & Hertzberger, L. O. (1995). Paradigms and laboratories in the core computer science curriculum: An overview. Special Interest Group Computer Science Education (SIGCSE) Bulletin, 27(4), 13 - 20.

Huck, S. W., Cormier, W. H., & Bounds, W. G. Jr. (1974). Reading statistics and research. New York: Harper & Row Publishers.

Impagliazzo, J., & Nagin, P. (1995). Computer science: A breadth-first approach with C. New York: Wiley.

Ivey, B. (1992). A case study of student learning in micro-computer based chemistry laboratory. Unpublished doctoral dissertation, Montana State University, Bozeman.

Jastrow, R. (1987). Towards an intelligent man. In J. Watson, (Ed.). Information Systems for Management. Plano: Business Publications, Inc.

Johnson, D. W., Johnson R. T., & Smith K. A. (1991). Cooperative learning: Increasing college faculty instructional productivity, The George Washington University, Washington, D.C.

Jung, J., & Brookshear J.G. (1994). Experiments in computer science (C Version). Redwood City, CA: The Benjamin/Cummings Publishing.

Kahn, K. (1996). Drawing on napkins, video-game animation, and other ways to program computers. Communications of ACM 39(8), 49-59.

Kernighan, B. W., & Ritchie. M. (1988). The C programming language. Englewood Cliff, NJ: Prentice Hall.

Kidwell, P.A., & Ceruzzi, P.E. (1994). Landmarks in digital computing. Washington, D.C.: Smithsonian Institute.

Knowles, M. S (1980), The modern practice of adult education: From pedagogy to andragogy (2nd ed.), New York: Cambridge Books.

Knowles, M. S., & Associates (1984), Andragogy in action--Applying modern principles of adult learning, San Francisco: Jossey-Bass Publishers.

Knowles, M. S. (1986). Using learning contract. San Francisco: Jossey-Bass Publishers.

Koffman, E. B. (1989). Pascal: Problem solving and program design, Reading, MA: Addison-Wesley Publishing Company, Inc.

Knuth, D. E. (1973). The art of computer programming. Reading, MA: Addison-Wesley Publishing.

Knuth, D.E. & Pardo, L.T. (1976). The early development of programming languages. International Research Conference on the History of Computing (pp. 197-264), Los Alamos Scientific Laboratories, New Mexico.

Leonard, J. R. (1991) Using A software engineering approach to CS 1: A comparative study of student performance, A Quarterly Publication of the Association for Computing Machinery Special Interest Group on Computer Science Education (SIGCSE Bulletin), 23(4), 23 - 26.

Levy, S. P. (1995). Computer languages usage in CS1: Survey results. Special Interest Group Computer Science Education (SIGCSE) Bulletin, 27(3), 21-26.

Lin, J. M., Wu, C. C., & Chiou G. F. (1996), Critical concepts in the development of courseware for CS closed laboratories, Conference on Integrating Technology into Computer Science Education (pp. 14-19). Barcelona, Spain.

Lodsdon, T. (1980). Computer and social controversy. Rockville: MD: Computer Science Press.

Lorenz, M. (1993). Object-oriented software development--A practical guide. Englewood Cliffs: Prentice Hall.

Mageau, T. (1990). Teaching and Learning On-line. Electronic Learning, 2, 26-30.

Marshall, L. (1995). Computers and Learning. Unpublished doctoral dissertation, Montana State University, Bozeman.

Mclave, S. (1986). Probability and statistics for engineers (2nd ed.). Boston: PWS Publising.

Merriam, S. B. (1988). Case study research in education: A qualitative approach. San Fransico: Jossey-Bass.

Merriam, S. B., & Caffarella, R. S. (1991). Learning in adulthood. San Fransico: Jossey-Bass.

Moreau, R. (1984). The computer comes of age: The people, The hardware, and the software. Cambridge: MIT Press.

Morris, J. M. (1992) The effect of an introductory computer course on the attitudes of older adults towards computers. Twenty-third SIGCSE Technical Symposium on Computer Science Education (pp. 72-75). Kansas City, Missouri.

Moyer, W. (1990). An interview with Myles Horton. In R. Fellenz & G. Conti (Eds.), Social environment and adult learning, Bozeman: Center for Adult Learning Research, Montana State University.

National Science Foundation (1992), America's future: A report of the Presidential Young Investigators Colloquium on U.S. Engineering, Mathematics, and Science Education for the Year 2010 and Beyond. Washington, D.C.: Directorate for Education and Human Resource.

Newstrom, J. W., & Scannel, E. E. (1980). Games trainers play: Experiential learning exercise. New York: McGraw-Hill Inc.

Paxton, J., Ross, R. J., & Starkey, J. D. (1993). An integrated, breadth-first computer science curriculum based on Computing Curriculum 1991. Twenty-fourth SIGCSE(Special Interest Group on Computer Science Education) technical symposium on computer science education (pp. 68-72), Indianapolis, Indiana.

Paxton, J., Ross, R. J., & Starkey, J. D. (1994). A methodology for teaching and integrated Computer Science Curriculum, Twenty-fifth SIGCSE(Special Interest Group on Computer Science Education) technical symposium on computer science education (pp. 1-5), Phoenix, Arizona.

Plato, (1970). Meno (G. Grube Trans.). Indianapolis: Hackett Publishing Company, Inc.

Prather, R. E. (1992). Computer Science in an undergraduate liberal arts and sciences setting, Special Interest Group Computer Science Education (SIGCSE) Bulletin, 24(2), 59-64.

Roberge, J. & Suriano, C. (1994) Using Laboratories to Teach Software Engineering Principles in the Introductory Computer Scienec Curriculum. Twenty-fifth SIGCSE(Special Interest Group on Computer Science Education) technical symposium on computer science education (pp. 106-110), Phoenix, Arizona.

Roger, C. R. (1969). Freedom to learn. Columbus: Charles E. Merrill.

Seaman, D. F., & Fellenz, R. A. (1989). Effective strategies for teaching adults. Columbus, Ohio: Merrill Publishing Company.

Sebesta, R. W. (1996). Concepts of programming languages. Reading, MA: Addison-Wesley Publishing Company.

Shaw, M. (1991). Informatics for a new century: computing education for the 1990s and beyond. Education and Computing, 7, 9-17.

Shiflet A.B. (1995). Problem solving in C including breadth and laboratories. St. Paul, MN: West Publishing.

Skinner, B. F. (1974). About behaviorism. New York: Alfred A. Knopf.

Starkey, J. D., & Ross, R. J. (1984). Fundamental programming with Pascal. St. Paul, MN: West Publishing Company.

Steinaker, N., & Bell, M. R. (January, 1975). A proposed taxonomy of educational objectives: The Experiential Domain, Educational Technology, 14 - 16.

Sullivan D. R. (1990). Computing today. Palo Alto, CA: Houghton Mifflin Company.

Thweatt, M (1994). CS1 closed lab vs. open lab experiment. Twenty-fifth SIGCSE(Special Interest Group on Computer Science Education) Technical Symposium on Computer Science Education (pp. 80-82). Phoenix, Arizona.

Tucker, A. B., Bernat, A. P., Bradley, W. J., Cupper, R. D., & Scragg, G. W. (1995). Fundamentals of computing I. New York; McGraw-Hill, Inc.

Tucker, A. B., & Garnick, D. K. (1991). Recent evolution of the introductory curriculum in computing. Education and Computing, 7, 43-60.

Weinberg, G. M. (1971). The psychology of computer programming. New York: Van Nostrand Reinhold.

White, M. A. (1988). The third learning revolution. Electronic Learning, 7(4), 6-7.

Winograd, T. (1983). Learning as a cognitive process--Syntax. Reading, MA: Addison Wesley.

APPENDICES

APPENDIX A

BEGINNING-OF-SEMESTER SURVEY

CS210 Introduction to Computer Science I **Fall 1996**

We, the faculty of the Computer Science Department, are interested in improving this course as much as we can. We appreciate it very much if you could just take about ten minutes to fill out the following questionnaires as honestly as you can. Please write legibly.

Name_____ **Major:** _____ **Age:** _____

Please circle one in each of the following questions:

1. **Gender:** Male Female

2. **Year in school:**

Freshman Sophomore Junior Senior Graduate

3. **Is this class required in your major?** Yes No

4. **Have you had programming experience?** Yes No

if yes, **please elaborate**

Language	**Number of years**	**School/Course**
_____	_____	_____
_____	_____	_____
_____	_____	_____

5. **Have you used a computer before this class?** Yes No

if yes, **what operating systems have you used?**

DOS OS2 Windows UNIX VMS unknown

others (list) _____

name all software packages you have used:

6. **Do you own a personal computer?** Yes No

if yes, **what?**

286 386 486 Pentium Apple Mac

others (list) _____

APPENDIX B

END-SEMESTER SURVEY

CS210 Introduction to Computer Science I
Course Evaluation

We, the faculty of the Computer Science Department, are interested in improving this course as much as we can. We appreciate it very much if you could just take about ten minutes to fill out the following questionnaires as honestly as you can. Please write legibly. No one will read any of this before the grades have turned in.

1. Rank the following items in the order of importance to learning in this class (the most important item should be ranked number 1, and so on)

_____ Lectures and text
_____ Programming Assignments
_____ Tests
_____ Labs
_____ Help from tutor and/or instructor

---

2. How can the lab. be improved?

---

3. How can the course be improved?

---

4. What feature(s) of this course should be kept?

APPENDIX C

LAB EXERCISE WITH RECORD AND EXPLAIN ACTIVITIES

CS210 Introduction to Computer Science I
Lab Report 6 *Logic and Truth Tables*

Name:_____ **Name:**_____

1. **Extract the following file from n:\\mtntsb\lab_kwan. The purpose of this program is obvious. Compile, link, and run the program. Record the results.**
To connect to drive n:
go to DOS shell
type in use n: \\mtntsb\lab_kwan
The file is in the subdirectory cs210

```
#include <stdio.h>
void main()
{ int operand1, operand2;

printf("\n\n\nTruth table of logical operation &&
(and)\n\n");
printf("operand 1    operand 2    operand 1 &&
operand2\n");
printf("-----------------------------------------\n")
;

operand 1 = 0; operand2 = 0;  /*  both false */
printf("%4d %12d %20d\n",operand1, operand2, operand1 &&
operand2);
operand 1 = 0; operand2 = 1;
printf("%4d %12d %20d\n",operand1, operand2, operand1 &&
operand2);
operand 1 = 1; operand2 = 0;
printf("%4d %12d %20d\n",operand1, operand2, operand1 &&
operand2);
operand 1 = 1; operand2 = 1;  /*  both true */
printf("%4d %12d %20d\n",operand1, operand2, operand1 &&
operand2);

printf("\n\n\n");

}
```

2. **Does the result correspond to the "ham and cheese" example in class? If yes, in what way?**

3. **Modify the program to output the truth table for the logical operator** `||`.

4. **Prove the DeMorgan's Theorem** `!(op1 || op2)` **is equivalent to** `!op1 && !op2` **by generating two truth tables. Your program should generate two tables as follow:**

Truth table of logical operation not(op1 or op2)

op1	op2	op1 || op2	!(op1 || op2)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

Truth table of logical operation (not op1) and (not op2)

op1	op2	!op1	!op2	!op1 && !op2
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

5. **turn in a hard copy of the program in step 3 and step 4 with test runs.**

APPENDIX D

LAB EXERCISE WITH EXPERIMENT AND DISCOVER ACTIVITIES

CS210 Introduction to Computer Science I
Lab Report 9 Sequential VS Binary

Name:_____ **Name:**_____

In this exercises in this laboratory, we add improvements to the program in Example 5.16 of Section 5.4. That program employed the function PlayGame to play a guessing game. After each addition, be sure to test the program. To make debugging easier, the program, which is the file LAB051.c on your disk, has a guessing range of 0 through 9. Copy this file onto your disk.

1. This exercise examines two techniques for making guesses. Play the game several times to get a feel for its action. Try each of the following methods for playing the games:

   a. Guess the number in order, 0, 1, 2, .., until hitting the target.
   b. Guess the middle number of the range each time until hitting the target. For example, for a range 0-9 with even number of choices, the first guess would be 4 or 5. Suppose we type 4, and the computer responds "Guess higher." Then our range is 5-9. With an odd number of choices, the middle is 7. The process continues until you find the number.

   The first method is called a **sequential search**, and the second is called a **binary search.** For each method, what is the least number of guesses you have to make? What is the most? Try the method you like best several times on the range from 0-99. What is the most number of guesses for each method? What is the most number of guesses for each method for the range 0-1022? Which method is faster for playing the game? Explain your answer, giving several examples.

APPENDIX E

LAB EXERCISE WITH DESIGN AND JUSTIFY ACTIVITIES

CS210 Introduction to Computer Science I
Lab Report 10

Name:_____ **Name:**_____

1. Design an algorithm to print out all possible outcomes of lotto ACM. Assume ACM uses a lottery for fund raising purposes and lotto ACM has ten balls labeled 'A' to 'J'. Three balls are drawn at random each week. Print all possible tickets as below with 8 per line and the total number of tickets at the end.

lotto ACM

ABC	ABD	ABE	ABF	ABG	ABH	ABI	ABJ
ACD	ACE	ACF	ACG	ACH	ACI	ACJ	ADE
ADF	ADG	ADH	ADI	ADJ	AEF	AEG	AEH
AEI	AEJ	AFG	AFH	AFI	AFJ	AGH	AGI
AGJ	AHI	AHJ	AIJ	BCD	BCE	BCF	BCG
BCH	BCI	BCJ	BDE	BDF	BDG	BDH	BDI
BDJ	BEF	BEG	BEH	BEI	BEJ	BFG	BFH
BFI	BFJ	BGH	BGI	BGJ	BHI	BHJ	BIJ
CDE	CDF	CDG	CDH	CDI	CDJ	CEF	CEG
CEH	CEI	CEJ	CFG	CFH	CFI	CFJ	CGH
CGI	CGJ	CHI	CHJ	CIJ	DEF	DEG	DEH
DEI	DEJ	DFG	DFH	DFI	DFJ	DGH	DGI
DGJ	DHI	DHJ	DIJ	EFG	EFH	EFI	EFJ
EGH	EGI	EGJ	EHI	EHJ	EIJ	FGH	FGI
FGJ	FHI	FHJ	FIJ	GHI	GHJ	GIJ	HIJ

There are 120 tickets.

2. Justify your design by a walk through.

3. Discuss the changes you would have to make to print all the tickets if lotto ACM has 45 balls labeled 1 to 45 and each drawing draws 6 balls.

***** If you test your idea with a program, DO NOT print the tickets unless you are going to plant several trees next Spring (consider the number of possible tickets!!!) and having the program won't give you any more points! If you really want to write the program, please just write down the number of possible tickets.***

MONTANA STATE UNIVERSITY LIBRARIES
3 1762 10299007 2