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

[illegible]
Head, Major Department

Approved for the College of Graduate Studies

1/15/98
Date

Joseph J. Fedock
Graduate Dean

STATEMENT OF PERMISSION TO USE

TABLE OF CONTENTS

TABLE OF CONTENTS--Continued

TABLE OF CONTENTS--Continued

TABLE OF CONTENTS--Continued

LIST OF TABLES

LIST OF FIGURES

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).

<u>Design and Justify</u>. 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)

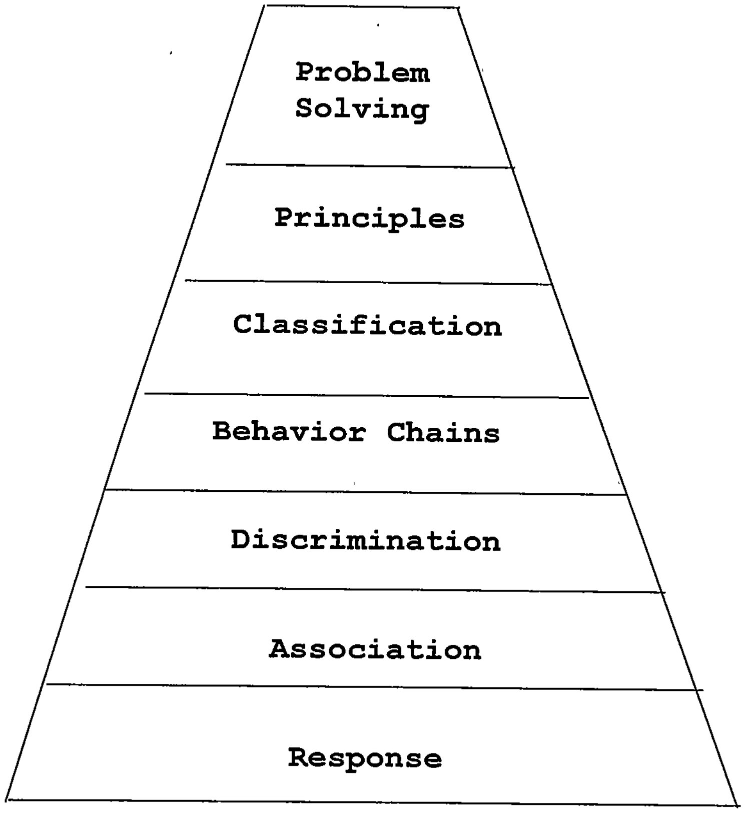
The typical classroom set-up with the traditional one-way transmission (Knowles, 1984, p. 15), most learners could hardly reach level 3. On the other hand, learning the syntax of a programming language can be viewed as knowledge based process (Winograd, 1983, pp. 2-29) which may only involve levels 1,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