Order Number 9323595

Conceptual models and individual cognitive learning styles in teaching recursion to novices

Wu, Cheng-Chih, Ph.D.

The University of Texas at Austin, 1993

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

CONCEPTUAL MODELS AND INDIVIDUAL COGNITIVE LEARNING STYLES IN TEACHING RECURSION TO NOVICES

by

CHENG-CHIH WU, B.ED., M.ED.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May, 1993

CONCEPTUAL MODELS AND INDIVIDUAL COGNITIVE LEARNING STYLES IN TEACHING RECURSION TO NOVICES

APPROVED BY DISSERTATION COMMITTEE:

Lowell J. Bethel

Nell Dale

[illegible]

Gordon Novak

James P. Barufaldi

To my wife, Shih-Ching

Acknowledgements

I would like to express my gratitude to the members of my dissertation committee: Dr. Lowell J. Bethel for his guidance on research methodologies and continuous encouragement; Dr. Nell B. Dale for her inspiration and strong support in carrying out the investigation; and Drs. James P. Barufaldi, John P. Huntsberger, and Gordon S. Novak Jr. for their unswerving support and helpful suggestions.

I would like to thank the participating students who provided invaluable information and the instructor, Suzy Gallagher, who gave her graciously assistance for this investigation. I am also grateful to Vicki Almstrum, Debra Burton, Dean Johnson, Xiang-Seng Lee, Jerry Norton, Quincy Spurlin, and Angel Syang for their professional advice and friendship during the course of the investigation. A special thank is due to Helen Kluna and Sonya Payne for their help during my stay in Texas.

Appreciation is extended to the Ministry of Education, Taiwan, R.O.C. for providing the scholarship and support for my doctoral studies in the U.S.A.

A very special thanks is due to my mother, Koan Wang Wu, and my parents-in-law, Pao-Hsiu Lin and Hsiu-Shiung Wang, for their profound support and encouragement throughout all these years. Finally, I would like to thank my wife, Shih-Ching, and son, Pei-Shin, whose love and support were essential for the completion of the dissertation.

CONCEPTUAL MODELS AND INDIVIDUAL COGNITIVE LEARNING STYLES IN TEACHING RECURSION TO NOVICES

Publication No.____________

Cheng-Chih Wu, B.Ed., M.Ed.
The University of Texas at Austin, 1993

Supervisors: Lowell J. Bethel and Nell B. Dale

This study investigated how different types of conceptual models and cognitive learning styles influence novice programmers when learning recursion. A pretest-posttest, 2 X 2 (conceptual models X learning styles) factorial experimental design was implemented in order to study the problem. Two hundred thirty-seven students enrolled in an introductory computer science course at a major southwest research university served as the subjects for this study. Subjects were randomly assigned to either an abstract model group or a concrete model group and the groups were of approximately equal size. Different conceptual models (abstract or concrete) were used to present recursion to the two model groups. Within each model group, subjects were identified as either an abstract learner or a concrete learner based on their scores on the scrambled Kolb's Learning-Style Inventory 1985. A posttest and two retention tests were administered after the treatment to compare students' performance in different groups. A pretest administered prior to the treatment was used to equate the variance caused by students' prior knowledge in the statistical analysis. The statistical procedure of two-way ANCOVA was employed to analyze all of the performance data.

The findings of this study are: Concrete conceptual models were better than abstract conceptual models in teaching recursion to novice programmers. However, the teaching effects weakened several weeks after classroom instruction. Novice programmers with abstract learning styles performed better than those with concrete learning styles when learning recursion. Finally, abstract learners did not necessarily benefit more from abstract conceptual models, and concrete learners did not necessarily benefit more from concrete conceptual models.

A replication study with a longer treatment period that covers more aspects of recursive programming is recommended for future research. Additional research needs to be conducted to better understand students' mental models of recursion. Furthermore, future research should investigate how the other dimension of Kolb's learning styles (*i.e.*, active-reflective) relates to the instructional methods provided. It is also recommended that the relationship between the characteristic of learning tasks (or domains) and the matching of learning styles with conceptual models be investigated.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 BACKGROUND

Recursion is basic to computer science, whether it is understood as a mathematical concept, a programming technique, or a way of problem-solving (McCracken, 1987). Understanding recursion is thought to be central in understanding complex data structures and program control (Rohl, 1984). Computer science educators have found that recursion is a very difficult concept for students to learn and teachers to teach (Ford, 1982, 1984; Henderson & Romero, 1989; Kurland & Pea, 1983; Widenbeck, 1989). Pirolli and Anderson (1985) argued that the lack of everyday analogies for recursion is what makes it so difficult to learn. Kurland and Pea (1983) found that students tend to develop an incorrect mental model of recursion. Many of the students had formed a mental model of recursion as a form of looping. Other studies (e.g., Bhuiyan, Greer, & McCalla, 1991; Kahney 1983) found this same incorrect mental model of recursion and other incorrect models as well.

A **mental model** is a conceptual representation of an abstract concept or a physical system that provides predictive and explanatory powers to a person in trying to understand the concept or the system and guides their interaction with it (Norman, 1983). The system that the person is learning or using is defined as the **target system. A conceptual model,** which is defined by teachers, scientists, or engineers, provides an appropriate representation of a target system, appropriate in the sense of being accurate, consistent, and complete (Norman, 1983). Understanding a system can be defined as having an accurate mental model of the system. Conceptual models are used as tools for the understanding or teaching of a system. It is the responsibility of teachers to develop conceptual models that will aid students in developing adequate and appropriate mental models.

Many conceptual models have been used in teaching recursion to novice programmers such as the *Russian Dolls* model, the *process tracing* model, and the *mathematical induction* model. It is hoped that these models will facilitate the learning of recursion by helping students develop an accurate mental model of recursion.

In addition to conceptual models, individual differences such as cognitive learning styles, cognitive abilities, and previous experiences with a similar system play a role in the mental model formation process of learning computer systems (Jagodzinski, 1983; Sein & Bostrom, 1989). There is evidence that individual cognitive learning styles are related to programming ability in novice programmers (Cavaiani, 1989; Merrienboer, 1988, 1990). However, the influences of cognitive learning styles in students' learning of recursion has not been studied.

This study is designed to investigate how conceptual models and cognitive learning styles influence novice programmers in learning recursion.

1.2 Statement of Problem

The problem with which this study is concerned is as follows: Which of two conceptual models (concrete or abstract) will best help novice programmers with different cognitive learning styles (concrete or abstract) to learn recursion?

1.3 Purposes of the Study

Computer science educators have found that recursion is a very difficult concept for students to learn. Part of the reasons may be because there are few analogies of recursion in students' everyday lives. Conceptual models are used as an analogy to aid students in building mental models of the target system. The purpose of this study is to investigate the effectiveness of conceptual models designed to help novice programmers gain an initial understanding of recursion. An initial understanding means having an accurate mental model of the concept. It serves as the basis for further learning of higher level skills.

Another purpose of this study is to study the effects of students' learning styles on their learning of recursion. It has been shown that individual differences, such as learning styles, have an effect on how people perceive and process information, but no studies have been done relating learning styles to the learning of recursion. The learning styles examined in this study are abstract vs. concrete learning styles.

The relationship between learning styles and conceptual models used to present information is less clear in the literature. To understand the interactive effects between the provided conceptual models and students' learning styles is the other purpose of this study.

1.4 Research Questions

The research questions for this study are:

1. Are concrete conceptual models better than abstract conceptual models in helping students to learn recursion?
2. Do students with an abstract learning style (i.e., *abstract learners*) outperform students with a concrete learning style (i.e., *concrete learners*) in learning recursion?
3. Do students with an abstract learning style learn recursion better when provided with abstract conceptual models?
4. Do students with a concrete learning style learn recursion better when provided with concrete conceptual models?

1.5 Rationale

1.5.1 Conceptual Models

Ausubel's (1978) theory of subsumption and Mayer's (1981) theory of assimilation provide theoretical explanations for the effectiveness of conceptual models. Both theories consider meaningful learning as a process of connecting new material to prior knowledge. When no prior knowledge is presented when learning a new domain, external assistance must be provided. Ausubel used *advance organizers* as such aids. The corresponding concept in teaching novice programmers is a conceptual model. Both theories propose that students can form a better mental model through the assistance of a conceptual model.

Gentner (1983) proposed a Structural Mapping Theory (SMT) to explain the process by which users make an analogy from a conceptual model to the target system. This theory views an analogy as a relational structure that applies to one domain (the "base") and can be effectively applied to another domain (the "target"). Conceptual models act as a "base" from which inferences can be made about the target system. An abstract conceptual model is the one that has an abstract base domain such as mathematical models. A *concrete* conceptual model has a more concrete base domain such as concrete objects. According to SMT, concrete and abstract models are actually opposite ends of the same continuum. They delineate the target system with varying degrees of concreteness. The basic difference between these two models lies in the concreteness of the objects in the base domain. Gentner believed that different models lead to a predictable difference in understanding of the target domain.

In the domain of teaching programming, Mayer (1979) and du Boulay et al. (1981) argued for the advantages of explaining the process which takes place within the black box (computers). They believed that a concrete conceptual model which showed the process of the system at an appropriate level of details would improve learning. This is called a *glass box* approach. However, Kurtz and

Kemeny (1985) proposed that programming should be taught to novices so that they do not have to aware of how the machine functions. Teaching programming from the concept of abstraction is a *black box* approach.

In summary, it can be concluded that a concrete model is a concrete analogy of a target system in terms of another system. It shows the internal process of the system at an appropriately detailed level. An abstract model is a synthetic representation of the underlying conceptual structure of a target system. The internal details of the system are hidden through abstraction.

Previous research (e.g., Bayman & Mayer, 1984; Kieras & Bovair; Rumelhart & Norman, 1981; Schlager & Ogden, 1986) have shown the effectiveness of using conceptual models in teaching/training computer systems. Several studies (Borgman, 1983/1984; Halasz, 1985) further concluded that the effects were significant especially in creative and complex tasks. However, which type of conceptual model (concrete or abstract) is more effective in teaching is inconclusive. Bennet (1984) and Sein, Bostrom, and Olfman (1987) found that, for simple tasks, subjects trained with concrete models performed better than those trained with abstract models, but for complex tasks, the effect was reversed. Schlager and Ogden (1986) and Sein (1988) found no significant difference between the two types of models.

Not many studies have been done in the field of programming using conceptual models. Mayer's series studies (1981, 1982, 1985, 1987, 1988) have provided experimental evidence that concrete model promotes learning. However, his series research did not explore more complex conceptual knowledge involved in large program segments such as the concept of a loop or the concept of a data structure. Nor did he compare the effects of different types of conceptual models. The present investigation studied the effects of both types of conceptual models in a more complex conceptual knowledge domain -- recursion.

1.5.2 Cognitive Learning Styles

Researchers in mental models (Norman, 1983, 1987; van der Veer & Felt, 1988) have pointed out that style of information processing (i.e., cognitive learning styles) was one of main individual difference features that affected the formation and acquisition of mental models. Individual styles of information processing not only result in preferences for different modes of presentation of learning materials and of analogies, but also lead to individual differences in the organization of semantic knowledge.

Kolb's experiential learning (Kolb, 1984) is a theory of cognitive learning styles. He believes that it is the combination of how people perceive and how they process information that forms the uniqueness of their own learning style, *i.e.*, the most comfortable and productive way to learn. More specifically, there are two main dimensions of the process by which people learn. The first is the way we perceive new information and is presented as a *concrete-abstract* continuum. In new situations, some people prefer to sense and feel their way (Concrete Experience) while others prefer to think their way through (Abstract Conceptualization). The second dimension, *active-reflective* continuum, is how we process new information. Some people prefer to jump in and try things (Active

Experimentation) while others prefer to process new information by reflecting on it (Reflective Observation).

According to the theory, the extremes of each dimension are mutually exclusive. If we try to simultaneously perceive new information, for example, by Concrete Experience and by Abstract Conceptualization, a conflict situation will arise. To resolve the conflict, each individual must choose how to perceive the new information and how to process it. Therefore, each individual develops a preference, i.e., a learning style, to perceive and process new information.

There appears to be some connection between the conceptual models and the concrete-abstract dimension of learning styles. Individuals with an abstract learning mode tend to discover the rules and structures inherent in an abstract model. These individuals take an analytical conceptual approach to learning. Individuals who prefer a concrete learning mode take an experiential-based approach to learning. Therefore, the concrete model seems more appropriate. There is evidence (Bostrom, Olfman, & Sein, 1987; Sein & Bostrom, 1989) that abstract learners benefit more from an abstract model and are hampered by a concrete model. Concrete learners, on the other hand, benefit more from a concrete model. In addition, both Sein and Bostrom (1989) and Zuboff (1988) found that abstract learners performed better than concrete learners on their experimental tasks.

1.5.3 Conceptual Models in Teaching Recursion

Many conceptual models have been used in introducing recursion. Five widely used models will be examined below. As for the relative concreteness of the models, the first three can be categorized as concrete models and the remaining two as abstract models. Other conceptual models can be found in Murnane (1991).

Russian Dolls (Bowman & Seagraves, 1985; Dale & Weems, 1991) A Russian Doll can be taken apart into many successively smaller dolls of the same shape. It displays the process of invoking a smaller size of itself (recursive case) and eventually the recursive process stops when the last doll does not contain another (base case).

Process Tracing (Dale & Weems, 1991; Koffman, 1992; Kruse, 1982) This approach focuses on tracing the process generated by recursive functions, that is, how recursive functions work. This model is clearly a concrete model, but the degree of concreteness may be varied depending on the method used in tracing the process.

Stack Simulation (Dale & Lily, 1991; Greer, 1987; Tenenbaum & Augenstein, 1986) Recursion is introduced in terms of computer architectures for execution of recursive programs. Calls to functions or procedures are traced with explicit reference to the system stack mechanism that is used in the Pascal implementation of recursion.

Mathematical Induction (Aho & Ullman, 1992; Ford, 1984; Henderson & Romero, 1989) This approach introduces recursion in terms of the mathematical basis for its correctness; that is, proof by induction.

Structure Template (Pirolli, 1985/1986a, 1986b) This model provides novice programmers with samples of recursive programs and describes the base cases and recursive cases. Solving a recursive problem is similar to filling in the slots of base case(s) and recursive case(s) in a structural template.

Pirolli (1985/1986a) found subjects receiving the structure template model learned to program their recursive functions in less time than did subjects receiving the process tracing model. The performance on the tasks between these two groups was not compared. Greer (1987) found no significant difference in students' performance with recursive tasks when they were taught with architecture-oriented (stack simulation), theory-oriented (mathematical induction), and task-performance-oriented (structure template) models. Neither researcher investigated the effects of individual differences and its interaction with the conceptual models.

1.6 Research Hypotheses

Eight hypotheses are developed in order to answer the research questions. Performance on recursive tasks measured by a *posttest* immediately after the treatment and two *retention* tests after two and six weeks of the treatment, will be used to test the hypotheses.

Since the concrete conceptual models provide a concrete base domain to infer recursion, novices tend to gain more understanding of recursion through these models. The following two hypotheses will be examined in order to answer research question 1.

H1: Students instructed in recursion with concrete conceptual models will outperform those instructed with abstract conceptual models on the *posttest* measure.

H2: Students instructed in recursion with concrete conceptual models will outperform those instructed with abstract conceptual models on the *retention* measure.

It is hypothesized that abstract learners rely on logical thinking and develop theories to solve problems. They may perform better in learning an abstract concept such as recursion. For research question 2, the hypotheses are:

H3: Abstract learners will outperform concrete learners on the *posttest* measure.

H4: Abstract learners will outperform concrete learners on the *retention* measure.

Students with an abstract learning style may easily adapt ideas when provided with abstract conceptual models, whereas students with a concrete learning style may learn better when provided with concrete conceptual models. The hypotheses for research question 3 are:

H5: Abstract learners perform better on the *posttest* measure when provided with abstract conceptual models as opposed to concrete conceptual models.

H6: Abstract learners perform better on the *retention* measure when provided with abstract conceptual models as opposed to concrete conceptual models.

And, the hypotheses for research question 4 are:

H7: Concrete learners perform better on the *posttest* measure when provided with concrete conceptual models as opposed to abstract conceptual models.

H8: Concrete learners perform better on the *retention* measure when provided with concrete conceptual models as opposed to abstract conceptual models.

1.7 Significance of the Study

Recursion is an important concept in computer science. Most computer science students have difficulty in understanding the mechanism of recursion and in writing recursive programs. Consequently, they are often frustrated and fail the following data structure and algorithm classes, which are fundamental cores of computer science and require recursion as a prerequisite.

This study is an attempt to find effective ways to teach recursion and, at the same time, consider individual differences such as cognitive learning styles. If we can reduce or resolve the difficulty of learning recursion through this study, it will be of great help to the field of computer science education.

1.8 Definition of Terms

Recursion is a mechanism for defining something in terms of a simpler version of itself (See Appendix A).

Performance in Recursion as considered in this investigation is the achievement in two related skills (1) to read and understand recursive programs, (2) to construct recursive programs, i.e., to generate the base case(s) and recursive case(s) for a problem.

A **Mental Model** is a conceptual representation of an abstract concept or physical system that provides predictive and explanatory powers to a person in trying to understand the concept or the system and guides their interaction with it. It is internal to a person (Norman, 1983).

A **Conceptual Model** is designed by teachers, scientists, or engineers. It provides an appropriate representation of a concept or a system (target system). It is external to a person (Norman, 1983). Conceptual models act as a "**base**" from which inferences can be made about the target system.

An **Abstract Conceptual Model** or **Abstract Model** is described as having an abstract base domain such as mathematical models in inferring a target system. The internal details of the target system are hidden through abstraction.

A **Concrete** (or Analogical) **Conceptual Model** or **Concrete Model** is described as having a more concrete base domain such as concrete objects in inferring the target system. It shows the internal process of the target system at an appropriately detailed level.

Cognitive Learning Styles or **Learning Styles** are the unique ways whereby an individual perceives and processes new information and are the means by which an individual prefers to learn (Kolb, 1984).

Concrete Learners are individuals with a Concrete Learning Style who prefer to sense and feel when learning. They perceive information in concrete form and use intuition (Kolb, 1984).

Abstract Learners are individuals with an Abstract Learning Style who prefer to think their way through when learning. They use reasoning and analytical skills to perceive information (Kolb, 1984).

1.9 Delimitations

This study is concerned with how novice programmers learn recursion. The subjects under investigation are students who enrolled in the first computer science course (CS 304P) at a major southwest research university. Most students in the course are novice programmers. Thus, they are the perfect sample for this study. The only restriction is that there is just one lecture session scheduled for recursion in the course. However, the concept of recursion can still be presented in one lecture session without losing its completeness.

Because of the time factor, the scope of recursion in this study will be limited to recursive *functions* with *simple variables*. Recursion with *structured variables* and using *procedures*, which involves more complicated context, will not be investigated in this study.

1.10 Overview of the Dissertation

This study investigates how conceptual models and individual cognitive learning styles influence novice programmers when learning recursion. The dissertation is organized as follows:

Chapter 1 is an introduction to the study. The chapter addresses the background and rationale of the study. The purpose, research questions, and hypotheses of the study are also developed and contained in this chapter.

Chapter 2 is a thorough review of related literature and research. The theoretical background of conceptual models and cognitive learning styles together with the research findings in these two fields are analyzed and described. Next, the problems and the conceptual models used in teaching recursion are discussed. The chapter concludes with a summary of related research findings.

Chapter 3 describes the methodology of the study. The sample, experimental design, procedures, instrumentation, and data analysis are described in detail. The reliability and validity of the instrumentation are further investigated based on the data collected from the pilot studies and the present investigation.

The results of two pilot studies prior to the current investigation are reported at the end of the chapter.

Chapter 4 presents the findings of the study. Data collected from the experiment is analyzed using computer statistical procedures. The results of hypotheses testing are presented, followed by a summary of the findings.

Chapter 5 presents a discussion of the results of the study and conclusions together with implications and recommendations for future research.

Chapter 2 Related Literature

2.1 Introduction

To understand a system (or a concept) means having accurate mental models of the system (or the concept). Conceptual models are designed as aids to help students to build accurate mental models of the system. In the consideration of building mental models we need to consider four different things: the *target system* to be learned, the *conceptual model* of the target system, the user's *mental model* of the target system, and the *scientist's conceptualization* of that mental model. Norman (1983) made a clear distinction of these four terms:

(1) *Target system.* The system that a person is learning or using, e.g., a computer system. The target system in this investigation is the recursion concept within the context of Pascal programming.

(2) *Conceptual models.* A conceptual model is invented to provide an appropriate representation of the target system, appropriate in the sense of being accurate, consistent, and complete. Conceptual models are invented by teachers, designers, scientists, or engineers.

(3) *Mental models.* The concept denotes the knowledge structure a person applies in interacting with the target system. These models need not be technically accurate (and usually are not), but they must be functional. This model evolves

during interaction with the target system, especially during the initial learning phase.

(4) *Scientist's conceptualization of the mental models.* This is the idea that the psychologist or researcher has about the mental models of a person in interacting with the target system. To figure out what models people actually have requires one to go to the individuals, to do psychological experimentation and observation.

In addition to conceptual models, many researchers (e.g., Jagodzinski, 1983; Sein & Bostrom, 1989; van der Veer & Felt, 1988) have proposed that individual differences, such as prior experience or learning styles, play a role in human mental model formation process. The kind of individual differences concerned in this investigation is individuals' cognitive learning styles.

This chapter will first review the theoretical background of mental models, including their definition and how they are built and used by a person. Next, the two factors conceptual models and learning styles which affect the formation of mental models, will be described and analyzed. Finally, the target system in this investigation, recursion, as well as the conceptual models used to teach it will be thoroughly reviewed.

2.2 Mental Models

2.2.1 Schemata and Mental Models

Researchers have always considered the problem of mental knowledge representation as a fundamental issue in understanding complex human behavior. In resent years a number of constructs have been developed to deal with the mental representation of complex phenomena: *frames* (Minsky, 1975), *scripts* (Schank & Abelson, 1977), *schemata* (Rumelhart, 1980), and *mental models* (Johnson-Laird, 1980,1983). Brewer (1987) argued that frames, scripts, and schemata are all examples of one general class of knowledge structure and referred it as 'schemata'. He concluded that schemata are unconscious mental structures that underlie the major aspects of human knowledge and skill. Schemata interact with incoming information to modify the generic information in the schemata and to produce instantiated schemata of the incoming information.

West, Farmer, and Wolff (1991, p. 7) reviewed related literature and defined schemata by including the following ideas: (a) schemata are mental data structures; (b) schemata represent our knowledge about objects, situations, events, self, sequences of actions, and natural categories; (c) schemata are like plays and scripts of plays; and (d) schemata are like theories. In other words, schemata are like packets or bundles in which the mind stores knowledge: They are patterns, structures, or scaffolds. Schemata are generalized units of knowledge or memory representations about a particular domain or concept.

A number of researchers have pointed out that schemata are inadequate to account for a wide range of phenomena. Human beings are capable of dealing with situations that do not involve old generic information. Thus, we can understand actions that we have never carried out before. Schemata theory cannot explain the situations that do not involve old generic information. In addition, schemata are limited to representations of knowledge and seldom explain how such representations are used in problem solving and learning environments (Borgman, 1983/1984; Sein, 1988). Schemata are mainly a static memory representation and cannot be used to *run* this representation to simulate a problem to arrive at a possible solution. The construct of mental models proposed by Johnson-Laird (1980, 1983) was introduced to deal with these problems. He emphasized that mental models are specific, not generic, representations and argued that they give rise to images (1983). The images can be manipulated in problem solving or learning situations and provide the dynamic aspects of the memory presentation.

Brewer (1987) argued that schemata and mental models do not really differ in terms of the specific/generic dimensions nor in the issue of imagery. He concluded that they are just two forms of memory representations. Schemata are precompiled generic knowledge structures, while mental models are specific knowledge structures that are constructed to represent a new situation through the use of generic knowledge of space, time, causality and human intentionality. Recent developments of Anderson's ACT* theory (Anderson, 1983), which is based on schemata theory, account for problem solving aspects of knowledge representation.

Researchers' views about the differences between schemata and mental models are diverse. But, it may be concluded that the theoretical root of mental models is schemata theory (Bennett, 1984). Mental models represent the progression of schemata from a static representation to a more dynamic one.

2.2.2 Definitions

The term Mental Model is attributed to Johnson-Laird (1980):

> A mental model represents a state of affairs and accordingly its structure is not arbitrary like that of a propositional representation, but plays a direct representational or analogical role. Its structure mirrors the relevant aspects of the corresponding state of affairs in the world. (p. 98)

Using this construct, Johnson-Laird has been able to provide an account for a wide variety of phenomena, such as comprehension of texts involving spatial descriptions and inferences derived from a particular mental model of a specific situation. Thus, mental models, like schemata, capture an important characteristic of human cognition. In its most generic definition, the term Mental Models can be applied to any mental event or, somewhat narrower, to any thought process. In this sense, one can have a mental model of one's own behavior, another person's behavior, or any information process mediated by people or machines (Carroll & Olson, 1987).

Many mental models research have been carried out in the modeling of physical systems or computer packages. Several descriptions of the term Mental

Models exist in the literature. Norman (1983) stated that it is the major underlying conceptual theme in the area that:

> In interacting with the environment, with others, and with the artifacts of technology, people form internal mental models of themselves and of the things with which they are interacting. These models provide predictive and explanatory power for understanding the interaction. (p. 7)

Bennett (1984) described a mental model as "... an individual's knowledge and/or beliefs about a particular domain which allows effective reasoning within that domain" (p. 12). Sein (1988) proposed that "Mental models are the users' understanding or knowledge of the system that serves as reasoning aids" (p. 36). Kieras and Bovair (1984) defined mental models as "some kind of understanding of how the device works in terms of its internal structure and processes" (p. 255). All the definitions maintained the representational features and reasoning power of mental models.

A more conclusive and explicit definition was provided by Carroll and Olson (1987):

> The user's mental model of a system is defined as a rich and elaborate structure, reflecting the user's understanding of what the system contains, how it works, and why it works that way. It can be conceived as knowledge about the system sufficient to permit the user to mentally try out actions before choosing one to execute. (p. 12)

2.2.3 "Running" Mental Models

An important feature of a mental model is that it can be "run" with trial, exploratory inputs and observed for its results. This dynamic nature of a mental model distinguishes it from being simply a plain memory presentation. This "running" feature is based on the imaginal properties of mental models provided in the literature. For example, de Kleer and Brown (1981) stated that mental models are generated "by running a qualitative simulation in the mind's eye" (p. 286). Collins (1985) stated that mental models "imply a conceptual representation that is qualitative, and that you can run in your mind's eye and see what happens" (p. 80).

Mental models are used during learning (e.g., using an analogy to begin to understand how a system works), in problem solving (e.g., performing a novel task), and when the user is attempting to rationalize or explain the system's behavior. In other words, such a representation must be capable of simulation; users should be able to 'run' this representation to derive a possible solution for the problem. Thus, the quality of mental models becomes crucial in the running of the models. Accurate mental models will result in effective learning and better performance in problem solving; whereas inaccurate mental models may result in ineffective learning and poor performance in problem solving.

2.2.4 Building Mental Models

As with other theories in cognitive science, the mental models of human beings are not directly observable. No research has been able to 'prove' the existence of mental models. Therefore, researchers can only try to infer what models users hold by observing users interacting with a system or by users' self-reports. Much of the work on the formation and development of mental models has begun with the premise that the user possesses a mental model of the system and has then explored the characteristics of that mental model. Researchers first state the behavioral outcomes that can be attributed to the existence of the hypothesized structures of the mental models. Next, in an experimental setting, observations can be made to determine whether the predicted behavior occurred. Behaviors suggested and examined include, for example, quality of performance and learning, nature of errors, and the sketching and drawing of procedures (Sein, 1988).

Norman provided a fundamental framework for this line of research. He (1983) modeled a person's mental model of a particular system by defining four concepts. Let the particular *target system* to be learned be called *t*. The *conceptual model* of *t* is *C(t)*. The conceptual model is designed as a tool for understanding and teaching the target system. The person's *mental model* of the system is defined as *M(t)*. And the *scientists' conceptualization* of a mental model is *C(M(t))*. While the mental model is unobservable, researchers are forced to work within scientists' conceptualization of a person's mental model, C(M(t)), and must perform experimentation to figure out what models the person actually has. According to him, mental models are naturally evolving models and are constrained by things, such as previous experience with similar systems and the structure of human information processing systems.

Van der Veer and Felt (1988) pointed out that the development of mental models is generally considered to be strongly based on analogies and prior knowledge related to the new situation. This process can be activated if the teacher refers to existing semantic knowledge and schemata. The analogies stated here appear to be the same idea as the conceptual models defined by Norman.

Based on Norman's model, Sein (1988) proposed a framework of the mental model formation process in learning a system. Figure 2.1 is the framework.

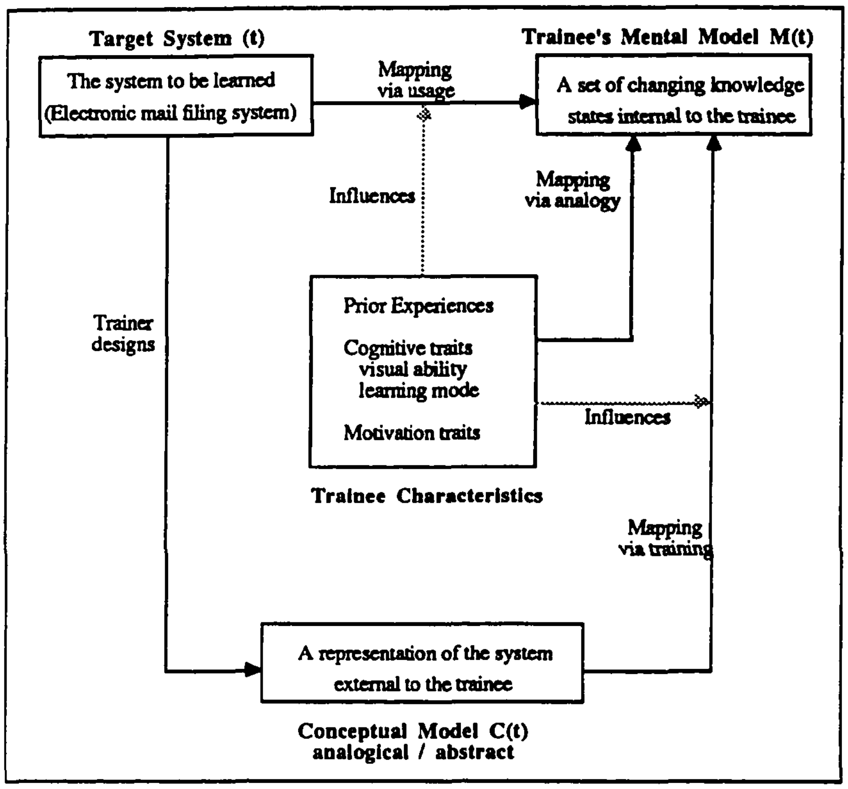


Figure 2.1 Mental Model Formation Process (from Sein, 1988)

The framework explicitly addresses the effects of individual differences in the model formation process. It postulates that a novice user can form a mental model of the system in the following three ways: (1) *Mapping via usage.* Users can acquire a mental model of the system by using it. (2) *Mapping via analogy*: Users can acquire a mental model of the system by drawing analogies from similar systems that are familiar to them. (3) *Mapping via training*: Users can acquire a mental model of the system through a conceptual model that is provided during training. A user can also form a mental model through multiple mappings. For novices learning a new system, they can be *trained with a conceptual model* of the system. This initial mental model can expand through *using* the system, and the user's prior relevant experiences (*analogy*) may interact with both the training and usage of the system.

It is obvious that conceptual models and individual differences, such as prior relevant experiences and information processing structures, play crucial roles in building mental models. The main concern of this investigation is how the conceptual models provided by teachers and the information processing structures (i.e., learning styles) possessed by students influence the formation of mental models. These two factors will be examined later in the next sections.

2.3 Conceptual Models

The previous section mentioned that people can be aided in building mental models of the system they are learning by being taught with a conceptual model. This section will describe the rationale of conceptual models as a teaching (or training) tool in detail, followed by a review of related research in the field.

Previous research (Gentner & Gentner, 1983; Kieras & Bovair, 1984; Mayer, 1981, 1988) has provided evidence that relevant conceptual models can facilitate students' learning and problem solving. These finding suggest that using conceptual models in instruction can enable analogical learning and cognition; in other words, when students acquire useful conceptual knowledge such as a conceptual model they can use this knowledge for learning and thinking about a new related domain (Norman, 1983). Rumelhart and Norman (1981) argued for the central role of analogical learning in cognitive theories by emphasizing *accretion*, "encoding new information in terms of existing schemata"; *tuning*, "slow modification and refinement of a schemata"; and *restructuring*, "creating new schemata" (pp. 335-336). The constructivists' view in science education has also recognized the importance of *accommodative learning*, corresponding to restructuring and tuning, as well as *assimilative learning*, corresponding to accretion (Wittrock, 1985).

Ausubel et al.'s *subsumption theory* (1978) demonstrated that students learn more when the new information being presented is preceded by a presentation at a higher order of abstraction which creates a conceptual framework into which the more specific information can be organized and anchored. This leads to the instructional strategy known as 'advance organizer'. Mayer's *assimilation theory* (1982) clearly demonstrated the effectiveness of using a conceptual model as an advance organizer. Both Ausubel and Mayers' theories provided the theoretical background of using conceptual models as teaching aids.

2.3.1 Ausubel's Subsumption Theory

Ausubel (1968, 1978) developed a theory of meaningful verbal learning called subsumption theory. The main ideas of the theory are concerned with how a person's prior knowledge and its organization determine learning. During meaningful learning the person organizes, or "subsumes" or incorporates, the new knowledge into old knowledge. New information can be better acquired and assimilated if it can be tied to knowledge already held in long-term memory. Within Ausubel's theory, the meaningful learning will occur if: (1) the learner holds previous relevant knowledge, (2) the material is logical, and (3) the learner intends to learn the material in a meaningful way. If these conditions are not met, students typically memorize in an unmeaningful way, make few attempts to incorporate the new material into their schemas, and usually forget quickly what they do learn.

The primary practical implication of subsumption theory has become the use of the *advance organizer*. The advance organizer is like a bridge, a linking of new information with something already known. The foundation is similarities between the old knowledge and the new. Without substantial similarities, the advance organizer is impossible. The advance organizer is introduced before a lesson or a unit of instruction; that is, before the main body of presentation. It provides the students with a structure for the new material. More than an ordinary introduction or transition, the advance organizer is based on students' prior knowledge.

Some researchers have found that advance organizers do improve learning, and some have found that advance organizers do not. For their meta-analysis, Luiten, Ames, and Ackerson (1980) examined 135 published and unpublished studies relating to the advance organizer. They concluded that there was a small but facilitative effect of the advance organizer on learning and memory. Further, this effect extended across ages of subjects and subject matter fields. In addition, they found that the effect increased with time; that is, when the instruction in the experiments extended to several days or weeks as compared to a few hours, the retention effects were stronger.

In Mayer's review (1979), he concluded that advance organizers should aid learning for difficult-to-assimilate materials, which are unfamiliar, technical, or otherwise difficult to relate to the learner's existing knowledge. He also pointed out that advance organizers had been most effectively used in mathematics and science topics. The target system of this investigation, recursion, meets all of these criteria.

The characteristics of a conceptual model are similar to those of an advance organizer. A conceptual model of a system is an analogy or a higher level abstraction of the system; it captures the essential elements of the system. In other words, similarities exist between the conceptual model and the system. The conceptual model is always introduced before the main body of presentation to give the learner an initial understanding of the system. And, conceptual models are generally designed for complex systems that are difficult to relate to prior relevant experiences. Therefore, a conceptual model can be considered as an effective advance organizer for teaching purposes.

2.3.2 Mayer's Assimilation Theory

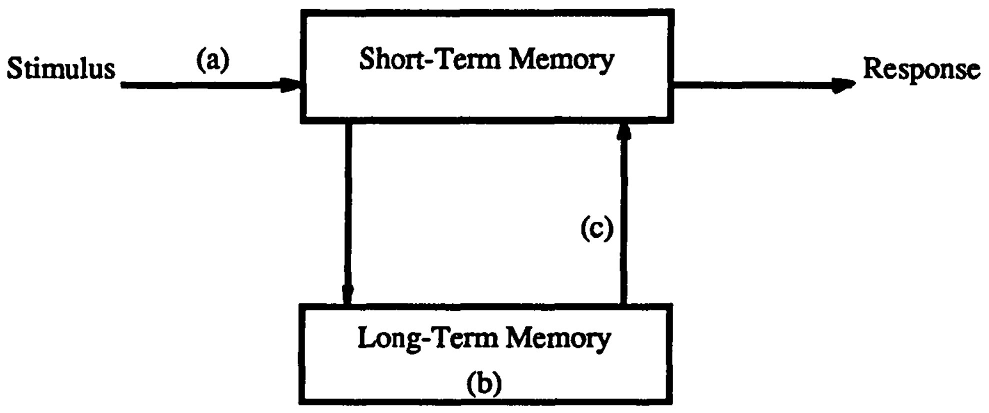
Mayer's assimilation theory (1981) provides a framework for the process of meaningful learning (or assimilation to schemata). *Meaningful learning* is viewed as the process in which the learner connects new material with knowledge that already exists in memory. The process, which is shown in Figure 2.2, occurs through the following three steps:

(1) *Reception.* The learner pays attention to the incoming information so that it reaches short-term memory (as indicated by arrow a).

(2) *Availability.* The learner possesses appropriate prerequisite knowledge in long-term memory to use in assimilating the new information (as indicated by point b).

(3) *Activation.* Finally, the learner must use this prerequisite knowledge during learning so that the new information may be connected with it (as indicated by arrow c).

If any of these steps is not met, meaningful learning can not occur, and the learner will be forced to memorize each piece of new information by rote.



Some information processing components of meaningful learning. Condition (a) is transfer of new information from outside to short-term memory. Condition (b) is availability of assimilative context in long-term memory. Condition (c) is activation and transfer of old knowledge from long-term memory to short-term memory.

Figure 2.2 Mayer's Assimilation Theory (from Mayer, 1981)

Mayer pointed out that novices generally have problems in step 2, availability, because of they lack domain-specific knowledge. One technique for improving their understanding of new information is to provide them with a framework that can be used for incorporating new information. This technique is aimed at ensuring availability of knowledge in long-term memory. The technique proposed by Mayer for providing the appropriate prerequisite knowledge is the use of conceptual models. For example, he used a pictorial conceptual model of a computer (Figure 2.3) to teach novices a BASIC-like programming language and the effect is significant (Mayer, 1981). The conceptual models used by him were concrete analogies for the computer system -- a scoreboard for memory, output pad for the output device, and ticket window for the input device. Mayer (1988) concluded that conceptual models presented before instruction tend to enhance the performance on transfer tasks which are creative or differ from those provided in the training, especially for weaker programmers. The conceptual models appear to serve as advance organizers for the new material to be learned.

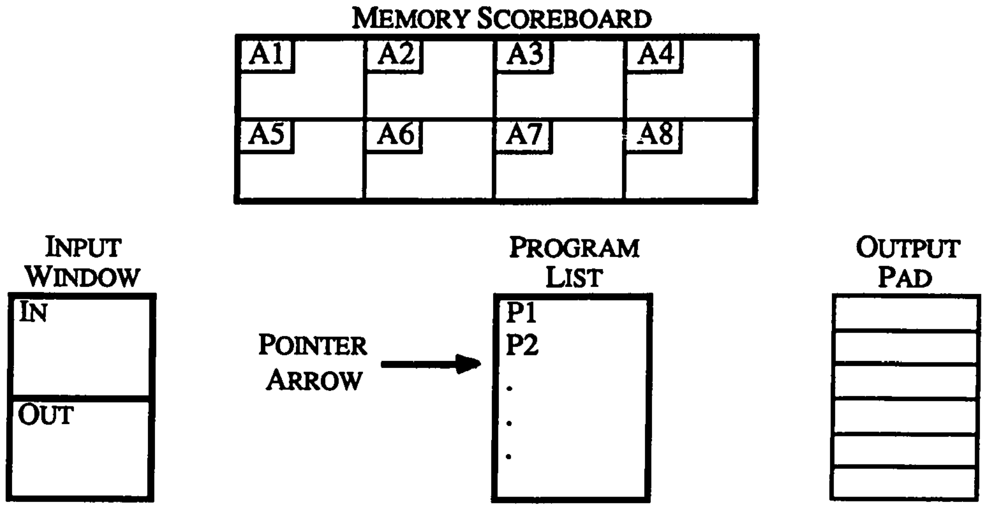


Figure 2.3 Mayer's Concrete Model of the Computer for a BASIC-like Language

In summary, both Ausubel and Mayers' theories propose that external aids, such as advance organizers and conceptual models, facilitate incorporation of new information into preexisting knowledge. These external aids provide the basis for forming an initial mental model of a target system.

2.3.3 Concrete and Abstract Conceptual Models

Several researchers (e.g., Bennett, 1984; Carroll & Olson, 1987; Young, 1983) have described different types of conceptual models used in mental model research. For example, Carroll and Olson (1987) conclude that there are four types of models called surrogates, metaphors, glass boxes, and network models. The kinds of conceptual models investigated in the present investigation were concrete and abstract models which were defined based on Gentner's Structure Mapping Theory (SMT) (1983) and du Boulay et al.s' Black Box vs. Glass Box approach (1981).

Structure Mapping Theory (SMT)

Gentner (1983) proposed the theory to explain the process by which users map from a conceptual model to the target system. The central idea of the theory is that an analogy is viewed as a relational structure that normally applies in one domain (the *base*) can be applied in another domain (the *target*). The basic assumptions of her theory are: (pp. 156-157)

1. Domains and situations are psychologically viewed as system of objects, object-attributes and relations between objects.
2. Knowledge is represented as propositional networks of nodes and predicates. The nodes represent concepts treated as wholes; the predicates applied to nodes express propositions about the concepts.
3. Predicates are of two types. *Attributes* are predicates taking one argument and *relations* are predicates taking two or more

> arguments. For example, if x and y are objects, COLLIDE (x, y) is a relation, while LARGE (x) is an attribute. First order predicates take objects as arguments, while higher-order predicates take propositions as arguments.

Gentner (1988) distinguished four kinds of mapping (or analogy) in terms of the number of object attributes and relational predicates being mapped from base to target:

1. *Literal Similarity*. Many object attributes and relational predicates are mapped from base to target.

2. *Analogy*. Only (at least mainly) relational predicates are mapped and few or no object attributes can be mapped from base to target. The base domain are concrete objects whose individual attributes must be left behind in the mapping.

3. *Relational Abstraction*. Abstract relational structures of a base domain are mapped. The object nodes of the base domain are generalized physical entities, rather than particular concrete objects. Predicates from the abstract base domain are mapped into the target domain; there are no nonmapped predicates.

4. *Mere-appearance Match*. Chiefly object attributes, but no or few relational predicates, are mapped from base to target.

Table 2.1 shows these four kinds of mapping and their corresponding examples. There are no strict distinction between the kinds of mappings. For instance, as shown in Table 2.1, there is no principal difference between *analogy* and *relational abstraction*. The latter is viewed as the analogy of a higher level.

Table 2.1 Kinds of Mapping in SMT

Mapping	No. of attributes mapped to target	No. of relations mapped to target	Example
Literal Similarity	Many	Many	The K5 solar system is like our solar system.
Analogy	Few	Many	The atom is like our solar system.
Relational Abstraction	Few*	Many	The atom is a central force system.
Mere-appearance	Many	Few	A sunflower looks like the sun.

* Relational abstraction differs from analogy and the other mappings in having few object attributes in both the base and target domain.

Relational abstraction is said to possess the most inferential power in the learning process. *Literal similarities* and *mere-appearance match* are considered as much less valuable in this respect, but access to the analogies is much more likely with these two kinds of mapping. *Analogy* is somewhere between literal similarities and relational abstraction. It facilitates the implementation of high inferential power because mainly relational structures are mapped. In addition, object attributes may ease access to analogies (Duit, 1991). Analogy and relational abstraction are considered most useful as teaching tools to bridge the prior and new knowledge.

According to the theory, analogy and relational abstraction are really opposite ends of the same continuum. They depict the target with varying degrees of concreteness. In fact, Gentner termed relational abstraction as abstract analogy. The basic difference of these two mappings lies in the concreteness of the objects in the base domain. For example, in Figure 2.1, the solar system is more concrete than the central force system. Thus, the conceptual models corresponding to analogy and relational abstraction mappings are termed as *concrete models* (or *analogical models* in some literature) and *abstract models*, respectively, according to the concreteness of their base domain.

A common example for these two models is given in teaching computer file systems. The analogy between a filing cabinet (base) and the computer file system (target) is clearly a concrete model. The basic structure and functions of a filing cabinet with labeled folders are mapped onto the structure and functions of a computer file system. The tree structure diagram (base) used to describe the concept of the hierarchical file system and the methods for traversing it (target) is considered as an abstract model.

Black Box vs. Glass Box

The idea of using conceptual models for introducing computer concepts was also developed by du Boulay et al. (1981). They suggested that too often teachers use a *black box* approach when teaching about computers. In this approach students are told not to be concerned about what happens inside the computer, but rather to look at each segment or function as a black box which has a set of inputs, a process, and a set of outputs. The process need only be presented in terms of the inputs and outputs without regard to what actually happens in the box. This approach may lead to programs that run, but it will hardly lead to an understanding of how a computer works.

Mayer (1979) argued for the advantages of explaining the process which takes place within the black box using an idealized set of parts. The parts need only be at a level of detail that will allow the processes to be explained. The level of detail for appropriate teaching is what he calls "transaction level". This is called as *glass box* approach. Du Boulay et al. (1981) suggested two important characteristics of programming languages for novices were needed in the approach: (1) *Simplicity*. It should consist of a small number of parts that interacts in ways that can be easily understood, possibly by analogy to other mechanisms with which the novice is more familiar. (2) *Visibility*. Novices should be able to imagine the selected parts and processes of the model in action.

A common glass box approach used in most introductory computer textbooks is to simulate the action of program statements on a conceptual model of the computer system. This approach offers the learner a view of the internal operation of the computer and the way the system reacts to programs. The level of detail must be sufficient to illustrate the concept to be learned, but should not introduce complexity that interferes with understanding the concept. For example, in describing the assignment statement, the "mailbox" analogy is often use to infer the concept of "computer memory location". The mailbox model is not as detailed as that of electronic flip-flops holding charges representing bits in the computer memory. Viewed at the electronic level, the concept of assigning a value to a variable is too complex. The greater level of detail would add more confusion rather than help in understanding the concept.

While revealing appropriate details may improve understanding, obscuring detail through *abstraction* may be equally important. Kurtz and Kemeny (1985) proposed that programming should be taught to novices so that they does not have to aware of how the machine functions. Teaching programming from the concept of abstraction represents a black box approach. Detail is kept from the learner's current consideration in order to facilitate learning. Procedural abstraction permits the programmer to consider a complete task as a single procedure. Similarly, data abstraction permits the programmer to consider data structures as entities, independent of machine level representation. It is believed that experts generally use abstraction as a means for simplifying the problem solving process.

The glass box approach shows the 'visible' process of the system in a simplified conceptual model to the learner. Mayer (1981) termed this kind of approach as a concrete model. While the black box approach which hides the details through abstraction is obviously an abstract model.

Conclusion

Based on the SMT and Glass Box vs. Black Box approach discussed above, it can be concluded that a concrete model is a concrete analogy of a target system in terms of another system. It shows the internal process of the system at an appropriately detailed level. Examples of concrete models are concrete objects (e.g., filing cabinet, and solar system in Table 2.1) or notational machines (termed by du Boulay et al., 1981) such as Mayer's pictorial model of a computer (See Figure 2.3). An abstract model is a synthetic representation of the underlying conceptual structure of a target system. The internal details of the system are hidden through abstraction. Examples of abstract models are abstract objects (e.g., tree structure diagram, and central force systems in Table 2.1) and logical or mathematical models.

The central theme of a conceptual model is the analogy between the model and the target system. Care needs to be taken in the use of analogies. The misapplication of an analogy is one of the most common mistakes made by novices. The problem arises when a learner tries to extract more structure or relationships from an analogy than is warranted (du Boulay, 1986).

Halasz and Moran (1982) suggested that *a concrete model (analogical model in his term) is effective for communicating complex concepts to novices when used as a literary metaphor* whose function is simply to illustrate some salient points of the target system, but it is dangerous when used as a way of reasoning about computer systems. They suggested that *reasoning is much better done with an abstract model.* The basic problem with a concrete model is that it attempts to represent a conceptual structure with familiar concepts that are inappropriate for reasoning about computer systems. Whereas, an abstract model directly presents the underlying conceptual structures of computer systems to the user, providing him/her with an appropriate basis to reason about the system.

However, Mayer views concrete models differently. He showed repeatedly the effectiveness of using concrete models. In his conclusion, Mayer (1982) stated that:

> When appropriate models are used, the learner seems to be able to assimilate each new statement to his or her image of the computer system. ... If the goal is to produce learners who will be able to come up with creative solutions to novel problems, then a concrete model early in learning is quite useful. (p. 26)

It seems both of them agree that **concrete models are more useful for novices learning complex system in the early learning stages.** But they have different views about how far the concrete models can be carried out in the learning process. Mayer believes that concrete models are helpful in the long run while Halasz and Moran disagree. They feel that only abstract models are useful in the long run. There are no general agreement about which type of conceptual model is better than another. The conclusions may vary because of the concreteness of the models (models may show different degree of concreteness), the timing of using the models, or the characteristics of the target system. Several research results in this field will be reviewed in the next section.

2.3.4 Related Research

The literature on the study of mental models is rich. Previous research has been mostly on the effects of conceptual models in building mental models of physical devices and abstract concepts. A comprehensive review is provided in Gentner and Stevens' book, *Mental Model* (1983). The review here will focus on research related to the effectiveness of conceptual models in teaching/training computer related knowledge.

Mayer's series of research (1981, 1982, 1985, 1987, 1988) has provided experimental evidence that the use of concrete conceptual models promotes learning of programming. For example, he (1981, 1982) provided students with a diagrammatic model which incorporated a variety of concrete metaphors (e.g., input as a ticket window and storage as a file cabinet). Students who were exposed to this model before studying a training manual were later able to perform better on both programming and recall tasks. He concluded that a concrete conceptual model which acts as an advance organizer will help novices come up with creative solutions to novel problems, especially for low ability programmers.

Rumelhart and Norman (1981) used a composite of three concrete models: a secretary metaphor, which was used to explain that commands can be interspersed with text input; a card file metaphor, which was used to describe the deletion of a single numbered line from a file; and a tape recorder metaphor, which was used to convey the need for explicit terminators in files. The performances were good for all the three models. They also found that there were several cases in which a subject would employ one of the models when another was appropriate. In the same domain, Foss, Rosson, and Smith (1982) provided students learning to use a text editor with a concrete conceptual model that used a file folder as the metaphor. They found that students who were provided with the concrete model learned more in less time.

Borgman (1983/1984) studied the effect of training novice users of a computerized information retrieval system with conceptual models. The experimental group was trained with a concrete conceptual model that described the retrieval system in terms of a library catalogue while the control group was trained procedurally in the mechanics of the system. The model trained group and the control group performed equally well on simple, procedural tasks, but the model trained group performed better on complex tasks that required extrapolation from the basic operations of the system.

Kieras and Bovair (1984) taught two group of students how to operate a simple device. One group learned a set of operating procedures for the device by rote, and the other learned a concrete conceptual model of the device (a block diagram representation) before receiving identical procedure training. The model group learned the procedures faster, retained them more accurately, executed them faster, and inferred the procedures more easily than did the rote group. They concluded that the useful how-it-works knowledge is the knowledge about the internal workings of the system that allows the user to infer exactly how to operate the device.

Bayman and Mayer (1984) trained calculator users with two different types of abstract conceptual models. They found the two model groups adopted very similar strategies that lead to accurate solutions of the experimental tasks. However, the control group, which had no model training, adopted random strategies and, were less accurate than the model trained groups. Halasz (1985) had similar findings. He taught students how to use a calculator using either a step-by-step action sequence to do standard calculations or instructions which included a verbal model of how the internal registers, windows, and stacks worked. They found that performance on standard tasks was identical for the two groups, but that the group that learned the model performed better on novel tasks.

Galletta (1985/1986) provided a group of novice users with a concrete model that described an electronic mail system in terms of a hotel telephone system. Their performance in using the mail system was compared against the performance of a control group who received procedural instruction. The concrete model group did not perform better than the control group.

Schlager and Odgen (1986) incorporated both a concrete model (conceptual model in his term) and a abstract model (procedural model in his term) in the training materials for teaching students how to form successful queries in a database. Three groups of subjects received either a concrete model, abstract model, or neither (the control group) in addition to basic instruction. The two model groups solved the problems faster than the control group, but did not differ from each other.

Bennett(1984) compared the effectiveness of two types of interface design for a computerized auditory database system. He found that the concretely trained group performed better than the abstractly trained group in simple tasks, but the effect was in the reverse for the complex tasks. Sein et al. (1987) had similar findings in training novices to use an electronic mail filing system.

Sein (1988) again investigated the effectiveness of two types of conceptual models (concrete and abstract models) in training novices using an electronic mail filing system. The concrete conceptual model represents the system in terms of an office steel filing cabinet, while the abstract conceptual model represents the system in terms of a hierarchical diagram. The performance was not significantly different between these two model groups. However, they observed significant interaction effects between the conceptual models used in training and individual differences -- learning style and visual ability. The individual difference effects were also observed in Sein et al. (1987).

Table 2.2 is the summary of the 12 studies reviewed. The first nine studies investigated the effectiveness of using conceptual models in teaching/training. Almost all of them supported the idea that the conceptual models were useful and better than the methods provided for the control group. Only Galletta's study (1985/1986) did not agree. Several of the studies (Mayer, 1981, 1982, etc.; Borgman, 1983/1984; Halasz, 1985) further concluded that the effects were significant in creative and complex tasks. Assuming the conceptual models were helpful, the last four studies in Table 2.2 compared the effects between two different types of conceptual models -- concrete and abstract. Their results were inconclusive. Bennett (1984) and Sein et al. (1987) found the effects were varied and depended on the complexity of the tasks. Schlager and Ogden (1986) and Sein (1988) found no significant difference between the two types of models. However the interaction effects between the types of conceptual models provided and individual differences were found in both Sein et al. (1987) and Sein's (1988) studies.

Table 2.2 Related Research in Conceptual Models

Study	Target Domain	Model Type Studied	Results
Mayer (1981, 1982, etc.)	Programming	Concrete	The model groups performed better than the control group, especially in creative and novel tasks.
Rumelhart & Norman (1981)	Text Editor	Concrete	The concrete models were useful.
Foss et al. (1982)	Text Editor	Concrete	The model group learned more in less time.
Borgman (1983/1984)	Information Retrieval	Concrete	The model group performed better on complex tasks.
Kieras & Bovair (1984)	Control System	Concrete	The model group performed better than the control group.
Bayman & Mayer (1984)	Calculator	Abstract	The model groups performed more accurately than the control group.
Halasz (1985)	Calculator	Abstract	The model group performed better on novel tasks.
Galletta (1985/1986)	Mail System	Concrete	The model group did not perform better than the control group.
Schlager & Ogden (1986)	Database Querying	Concrete/ Abstract	The two model groups perform better than the control group. No difference was found between the two model groups.
Bennett (1984)	Auditory Database	Concrete/ Abstract	The concrete model group performed better than the abstract model group in simple tasks. The effect is reversed in complex tasks.
Sein et al. (1987)	Mail Filing System	Concrete/ Abstract	Same as above.
Sein (1988)	Mail Filing System	Concrete/ Abstract	No difference were found between the two model groups.

Of all the studies reviewed, only Mayer's studies investigated the effects of conceptual models in programming domain. He repeated showed the effectiveness of concrete models in teaching novices programming. However, Mayer's series research did not explore more complex conceptual knowledge involved in large program segments such as the concept of a loop or the concept of a data structure. Neither did he investigate the effects of abstract models. The present investigation studied the effects of both types of models and individual differences, which was similar to Sein's study, but the target domain was in a more complex conceptual knowledge domain -- recursive programming. Learning styles, an important aspect of individual differences, is reviewed in the next section.

2.4 Cognitive Learning Styles

Individual differences among students present a pervasive and profound problem to educators. Students of any age will differ from one another in various intellectual and psychomotor abilities and skills, in both general and specialized prior knowledge, in interests and motives, and in personal styles of thought and work during learning. These differences, in turn, appear directly related to differences in the students' learning process. This implies that individual differences somehow condition students' readiness to profit from the particular instructional environments (e.g., conceptual models) provided. Learning is viewed as a process of mental model formation. Therefore, individual differences in conjunction with conceptual models should play a important role in the formation of mental models.

In his review of present knowledge about individual differences, Snow (1986) stated that cognitive learning styles were one of several personal constructs that deserve to be further explored. This investigation will explore the effects of learning styles in aiding the formation of mental models.

2.4.1 Learning Styles Theory

> Learning styles are characteristic cognitive, affective, and physiological behaviors that serve as relatively stable indicators of how learners perceive, interact with, and respond to the learning environment. (NASSP, 1979, p. 4)

Learning styles and cognitive styles have often been used synonymously in the literature although they are not the same. Learning styles, in fact, is the broad term which includes three dimensions: cognitive, affective, and physiological styles.

Cognitive styles are information processing habits representing the learner's typical mode of perceiving, thinking, problem solving, and remembering (Messick, 1976). Each learner has preferred ways of perception, organization, and retention that are distinctive and consistent. The difference between cognitive styles and general cognitive abilities are as follows: Styles focus on "how I learn" and abilities focus on "what I learn"; style is bipolar or on a continuum; abilities are unipolar or measured with a single score. Examples of cognitive styles are reflection vs. impulsivity and field independence vs. field dependence (analytical as opposed to non-analytical way of experiencing the environment).

Affective styles involve personality and emotional characteristics related to areas such as persistence, locus of control, responsibility, motivation, and peer interaction. Do you prefer working by yourself or with peers? How do you respond to verbal or token reinforcement? Examples of affective styles are level of anxiety and competition vs. cooperation.

Physiological styles are biologically-based modes of response that are founded on sex-related differences and personal nutrition and health. Are you a morning, afternoon, or night person? Do you need frequent breaks? Examples of physiological styles are time-of-day rhythms and heath-related behavior. (Keefe, 1987)

Like intelligence or general ability, learning styles come as a result of our heredity, experiences, and environment (Kolb, 1984). They are a result of nature and nurture (Gregorc, 1984). While learning styles are considered to be consistent patterns of behavior and are relatively stable traits over time, they can be modified with age and experience. For example, with maturation, learning styles tend to move to greater abstraction and tend to become more analytical and reflective. Unlike intelligence or general ability, however, styles are value-free (Messiah, 1976; McCarthy, 1980). No style is better than another. Nor is learning by doing necessarily better than learning by listening. These styles are simply different.

However, some styles may be more effective than others in certain situations. Thus, when an individual learns, the style may be unique to the task or it may duplicate a previous experience. McCarthy (1980) found that individuals who preferred learning styles involving listening to lectures are usually accommodated in traditional classrooms. Carrier, Williams, and Dalgaard (1988) found that learning styles were a prediction of students' confidence and perception in their notetaking skills in a college economics course.

The instructional implication for teachers is that the more we know about students' learning styles, the more effective and efficient will be the teaching and learning process. Reiff (1992) states that there are several reasons for teachers to learn about students' learning styles: (1) reducing frustration for students and teachers, (2) improving students' self-concept and achievement, (3) helping teachers to plan and manage more appropriate lessons, (4) increasing variability and flexibility of students' learning styles, and (5) improving communication between teachers and students. The relationship between students' learning styles and the instruction provided, i.e. (3) and (5), is the main concern of this investigation.

The vast majority of research on personality-related learning variables has been in the area of cognitive styles (Keefe, 1987). Researchers in mental models (Norman, 1983, 1987; van der Veer & Felt, 1988) have pointed out that style of information processing (i.e., cognitive styles) and prior knowledge were the two main individual difference features that affected the formation and acquisition of mental models. Individual styles of information processing not only result in preferences for different modes of presentation of learning materials and of analogies, but also lead to individual differences in the organization of semantic knowledge. This investigation focuses on the cognitive aspect of individuals' learning styles.

2.4.2 Kolb's Experiential Learning

In order to measure cognitive learning styles in this investigation, Kolb's Learning-Style Inventory 1985 (LSI-1985) was chosen. LSI-1985 is based on Kolb's experiential theory which views learning as a discovery process that incorporates the characteristics of problem solving and learning (Kolb, 1984, 1985). Kolb believes that it is the combination of how people perceive and how they process information that forms the uniqueness of their own learning style, i.e., the most comfortable and productive way for them to learn. More specifically, there are two main dimensions of the learning process by which people learn.

Concrete-Abstract dimension -- *How people perceive new information.* In new situations, some people prefer to *sense* and *feel* their way, while others prefer to *think* their way through. Those who sense and feel tend to rely on CONCRETE EXPERIENCE (CE). They perceive information in concrete form and use intuition. Others who think their way through focus more on symbolic representation or ABSTRACT CONCEPTUALIZATION (AC) of new information. They use reasoning and analytical skills to perceive information.

Active-Reflective dimension -- *How people process what they perceive.* Some people, who favor REFLECTIVE OBSERVATION (RO), watch or reflect on their experiences while others become active and are doers involved in ACTIVE EXPERIMENTATION (AE).

Kolb views AC and CE as being the opposite ends of a continuum of abstract conceptualization of information in learning. In other words, abstract conceptualizers (abstract learners) have an opposite learning mode to those who favor concrete experience (concrete learners). The same case is with the other continuum, i.e., active experimenters and reflective observers. Kolb has defined four learning styles (Figure 2.4) corresponding to each combination of preferred ways to perceive and to process new information. Four learning styles are defined based on a multi-dimensional model. The theory's learning mode dimensions can also be categorized under the single learning style continuum model. For example, along the AC-CE dimension, an individual can be categorized as either an abstract learner or concrete learner. The same case is with the AE-RO dimension.

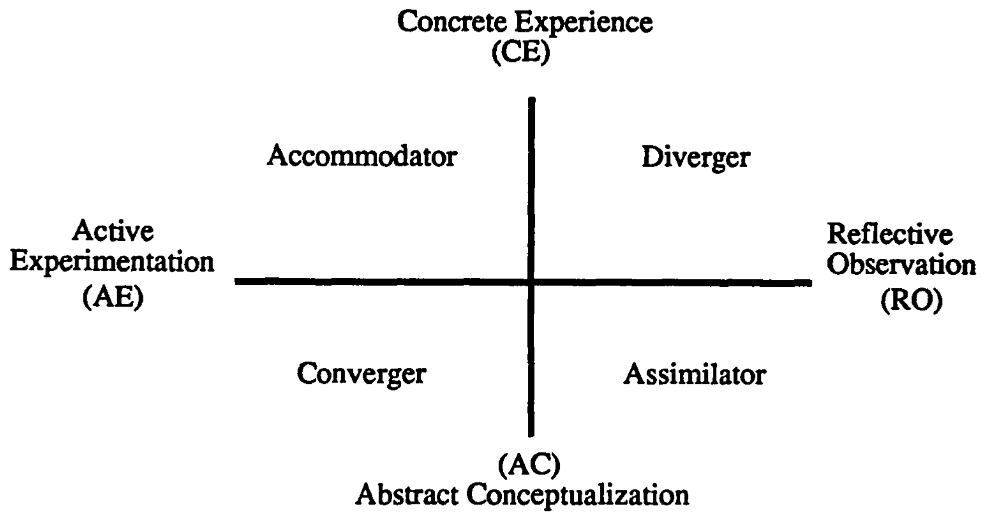


Figure 2.4 Kolb's Four Learning Styles

Kolb argues that because of the experiential nature of learning, different learning situations are necessarily different experiences. An individual may prefer one style in one situation and a different style in another. However, even if an individual's learning style varies with the situation, it will remain constant within a particular context. For example, while learning programming, an individual's learning style is likely to remain the same. Moreover, although an abstract learner may choose to employ concrete experience in a situation, he or she is more likely to be "less concrete" than someone who prefers concrete experiences in the same situation. LSI is therefore a relative measure of difference among individuals.

Kolb's theory has been widely applied in many research studies (e.g., Atkiston, Murrell, & Winters, 1990; Geiger, 1991; Pinto & Geiger, 1991; Reading-Brown & Hayden, 1989) for pedagogical purposes. His Learning-Style Inventory is considered better than other similar instruments in learning styles research (Karrer, 1988). The reliability and validity issues of the LSI-1985 will be discussed in the Instrumentation section of Chapter 3.

2.4.3 Learning Styles and Conceptual Models

Kolb's theory predicts that students with different learning styles respond differently to various teaching methods and that instructional strategies should match the learning styles of students. Some students may grasp abstract concepts readily while others need concrete imagery to learn. Individuals with different learning styles tend to learn differently from different teaching methods (Catalanello & Bremenstuhl, 1978; McCarthy, 1980).

There appears to be some connection between the conceptual models provided in instruction and the abstract-concrete (AC-CE) dimension of Kolb's learning styles. Individuals with an abstract learning style tend to discover the rules and structures inherent in an abstract model. These individuals take an analytical conceptual approach when learning; they rely heavily on systematic planning and develop theories and ideas to solve problems. Thus, an abstract model seems more helpful for abstract learners in the learning process.

On the other end of the continuum are individuals who prefer a concrete learning style. They take an experiential-based approach when learning and tend to rely more on their feeling and experiences than on a systematic approach to problems and situations. An abstract model is highly unlikely to be part of their relevant experience. Whereas, a concrete model seems very likely to be so. Therefore, a concrete model is more appropriate for concrete learners.

There is evidence (Bostrom et al. 1987; Sein & Bostrom, 1989) that abstract learners benefit more from an abstract model and are hampered by the concrete model. Concrete learners, on the other hand, benefit more from a concrete model.

The active-reflective (AE-RO) dimension of Kolb's learning styles deals with active involvement aspects in learning and is less related to any interaction with the provided conceptual models. This dimension was not investigated further in this study.

2.4.4 Related Research

Many recent studies attempt to correlate individuals' cognitive learning styles with the learning in *computer related domains*. For example, van der Veer and Felt (1988) investigated students' preferred style of representation of information (image or abstract representation) in learning an office system. No relation between the styles and mental models could be detected because of the small sample size (N = 10). However, they concluded that the learning styles as measured on the dimension of imager/nonimager seems to be relevant to the resulting mental model, especially regarding correctness and completeness.

Davidson, Savenye, and Orr (1992) studied the relationship among learning styles (as defined in Gregorc, 1984) and performance measures for computer concepts and application skills. Students who prefer the world of abstraction and symbols (abstract learners) achieved significantly higher in total course points; while those students who experience the world of reality through their imagination and feelings (concrete learners) earned significantly lower total course points. The major difference was found in programming related activities.

Several studies relate the performance in *programming* to cognitive learning styles. For instance, Corman (1986) surveyed the correlation between students' learning styles (as defined by Kolb, 1985) and their success in an introductory COBOL course. He found no significant correlation between students' learning styles and their success in the introductory programming course.

However, Cavaiani (1989) and van Merrienboer (1988, 1990) found cognitive learning styles did play a role in students' learning programming. Cavaiani (1989) investigated the influence of cognitive styles (field dependence-independence, as measured by Group Embedded Figure Test, GEFT) on the component programming skill of debugging. The results showed that individuals possessing a global cognitive style will be at a disadvantage when involved in the cognitive task of program comprehension and debugging. And, the analytic problem solvers perform better on diagnostic tasks than global (non-analytic) problem solvers.

Van Merrienboer (1988) conducted a study to explore the relationship between the cognitive style (reflection-impulsivity, as measured by Matching Familiar Figures Test, MFFT) and achievement in an introductory programming course. Reflectives were found to be superior to impulsives in their scores on a program comprehension test. No significance difference were found in scores on factual knowledge of programming language features and syntax. In his other research (1990), he found reflectives profit more from an instructional strategy that emphasizes the completion of existing programs, and impulsives from the generation of new programs.

Sein and Bostrom (1989) examined the influence of learning styles on the efficacy of conceptual models in learning a mail filing system. They found that abstract learners benefit more from an abstract model but are hampered by the concrete model. Concrete learners, on the other hand, benefit more from a concrete model. In the Bostrom et al.'s (1987) study, one of their four experiments provided the same findings, but three of them did not show any significant interaction effect.

In general, both Bostrom et al. (1987) and Sein and Bostrom (1989) found that abstract learners performed better than concrete learners on their experimental tasks and Zuboff (1988) had similar findings. Zuboff conducted an exhaustive study of factory workers adapting to computer-based control system in manufacturing machinery. She concluded that successful workers needed to develop abstract-thinking skills in order to develop accurate mental models of the system and to use their model. That was they needed to think through a process or problem rather than just physically acting on it.

Most of the studies reviewed suggests that there are relationships between individuals' cognitive learning styles and their performance in learning computing systems. The abstract or analytical learners tend to perform better than concrete or non-analytic learners in programming tasks. However, the interaction effects between students' learning styles and the conceptual models provided are inconclusive.

2.5 TEACHING RECURSION

There are many complex concepts that students encounter when learning computer science. Recursion, a powerful and elegant control structure of many computer languages, represents one of these fundamental and complex computer science concepts (Ford, 1982; Martin, 1985). Understanding recursion is thought to be central in a more complete understanding of complex data structures and program control (Rohl, 1984). McCracken (1987) argued that recursion is not an advanced topic, but rather "recursion is fundamental in computer science, whether understood as a mathematical concept, a programming technique, a way of expressing an algorithm, or a problem-solving approach" (p. 3). Recursion is also regarded as an exciting and powerful concept that can invoke a feeling of playing with infinity (Pampert, 1980; Turkle, 1985).

In spite of the importance of recursion, computer science educators have found that recursion is a very difficult concept for students to learn and teachers to teach (Anderson, Farrell, and Sauers, 1984; Ford, 1982, 1984; Henderson & Romero, 1989). It appears that students' response to recursion is often avoidance (Kurland & Pea, 1983; Widenbeck, 1989).

2.5.1 Problems in Learning Recursion

Recursion is used as a control structure in computer programs. It provides a means for repeating a process, similar to a loop construct. Block-structured languages such as Pascal, usually implement recursion by repeating execution of a function or procedure. A function or procedure can suspend its own action temporally, restart itself possibly with new input data, and when finished resume the suspended action and continue to completion. A recursive function or procedure is characterized by a call to itself. A detail description of recursion is located in Appendix A.

What makes recursion so difficult to learn? It may be due to three factors: lack of everyday analogies, limitation of human's short-term memory (or working memory), and complexity involved in recursive algorithms (Er, 1984; Pirolli, 1986b; Pirolli & Anderson, 1985).

When students learn programming concepts, they already have a lot of knowledge about how to perform everyday activities and how to use language. The analogies they make from their previous experiences probably help them to get some initial understanding of the programming concept (Bonar & Soloway, 1985). In the case of recursion, the concept is usually completely novel for novices and there are very few everyday analogies that exists. Those which are frequently used analogies, such as seeing one's own image reflected in a row of mirrors, convey the idea of infinity associated with the recursion, but do not convey the idea of recursion as a series of processes for solving a problem. Pirolli and Anderson (1985) argued that the lack of everyday analogies is what makes recursion so difficult to learn.

The problem in learning recursion may be also indirectly related to the general difficulty people have with executing recursive mental procedures. For example, people find it difficult to understand recursive linguistic structure such as *The boy is thinking that the girl is thinking of him thinking of her* (Eliot, Lovell, Dayton, & McGrady, 1979). It seems that the human mind cannot suspend one process, perform a recursive calculation, and return to the original suspended process. Such processing requires that partial results be maintained and distinguished in memory. It is a well-known psychological fact that the human short-term memory has a capacity limited to about seven "chunks" or items. Without external aids, the short-term memory simply can not cope with the huge amount of information generated by a recursive algorithm (Er, 1984; Pirolli, 1986b).

Er (1984) argued that it is not because the concept of recursion is difficult to grasp, but because recursive algorithms, by definition, are implemented with procedures or functions that rely on parameter passing mechanisms such as call-by-value or call-by-reference, together with global and local variables, to achieve their effects. Therefore, to comprehend recursive algorithms, one needs to understand three different aspects of computer programming: recursion, parameter passing mechanisms, and global versus local variables, all of which complicate the learning of the recursion concept itself.

Previous research has found that students tended spontaneously to develop an incorrect mental model of recursion. The most popular misconception was to represent recursion as a loop structure that iterates through a problem. Iteration, theoretically, a special case of recursion, is often introduced to students prior to recursion in an introductory programming course. Kurland and Pea (1983) collected think-aloud protocols and hand simulations from students as they tried to understand recursive programs in LOGO. Many of the students formed a mental model of recursion as a form of looping, rather than seeing it as the suspension of the current process and the passing of control to a completely new version of the same process. There are different kinds of recursion, and the incorrect mental model of recursion as looping may sometimes lead to correct results in some kind of recursion (i.e., tail-recursion). Because it did not always lead to incorrect results, the fault in the mental models was difficult for students to identify. Other researchers (Anzai & Uesato, 1982; Bhuiyan, Greer, & McCalla, 1991; Greer, 1987; Kahney 1983) found this same incorrect mental model of recursion and other incorrect models as well.

However, in general, research comparing the acquisition of recursive and iterative procedures has shown that novices are able to handle recursive concepts once they have learned iteration, but not vice versa (Anzai & Uesato, 1982; Kessler & Anderson, 1986; Wiedenbeck, 1989). Iteration is a common phenomenon in our everyday lives. The availability of real world analogies facilitates the development of a mental model of control flow in iteration. This mental model can serve as the basis for understanding recursion. Recursion, on the other hand, is a more complex mechanism and there are no good analogies that novices can draw on in formulating a mental model. Kessler and Anderson (1986) concluded that the reason for novices' difficulty in understanding flow of control in programming lies in their inability to develop adequate mental models of the task. Therefore, learning iteration prior to learning recursion is not the main argument for the development of incorrect loop mental model.

Thus, how to help novice students to develop an adequate mental model of recursion is the critical factor in teaching recursion. In a domain such as recursion, which is complex and does not have everyday analogies, using conceptual models as advance organizers appears to be an appropriate approach in instruction.

2.5.2 Models in Teaching Recursion

In teaching recursion, the ability to read and understand recursive programs and the ability to write a recursive algorithm to solve a specific a problem are generally considered the two basic skills introductory students must learn as reflected in the introductory computer science textbooks (e.g., Aho & Ullman, 1992; Dale & Lilly, 1991; Dale & Weems, 1991; Koffman, 1992). These two skills actually represent two stages of learning recursion: first, knowing what recursion is and how it works and then learning how to construct a recursive algorithm. Based on these two basic skills, researchers in this field have different perspectives of how to teach recursion. Some (Bowman & Seagraves, 1985; Lee & Mitchell, 1985; Murnane, 1991) emphasize the importance of knowing the flow of control of recursion and make less effort to explain how to derive a recursive solution; they prefer a glass box approach (concrete model) to demonstrate how recursion works. Some (Ford, 1984; Henderson & Romero, 1989; Pirolli, 1985/1986a) consider that the implementation details are trivial and it is better to explain this computer science concept in more abstract ways. They prefer a black box approach (abstract model) in addressing how to design a recursive algorithm.

The following are several conceptual models that are widely used in the field. The first three can be categorized as concrete models, and the remaining two as abstract models. The categorization is based on the relative concreteness of their base domain (as defined in SMT) and the detailed level the internal process shown in the models. Other models can be found in Lee and Mitchell (1985) and Murnane's (1991) review.

Russian Dolls. (Bowman & Seagraves, 1985; Dale & Weems, 1991) A Russian Doll can be taken apart into many successively smaller dolls of the same shape. It displays the process of invoking a smaller size of itself (recursive case) and eventually the recursive process stops when the last doll does not contain another (base case). This is considered a concrete model of recursion and better used as a literary metaphor (Halasz & Moran, 1982), whose function is simply to make a point in transferring the ideas of recursion, and not as tools for reasoning. Once the point is made the metaphor can be discarded.

Process Tracing. (Dale & Weems, 1991; Helman & Veroff, 1986; Koffman, 1992; Kruse, 1982) Process tracing focuses on tracing the process generated by recursive functions; that is, modeling the mechanism of recursion from the perspective of the scopes of procedures. This model is clearly a concrete model, but the degree of concreteness may be varied depending on the method used in tracing the process. The *block tracing diagram* (Appendix F) method which traces the flow of control and data among recursive calls by a block diagram, is more concrete than the *mathematical tracing* method. A mathematical tracing method, on the other hand, traces the recursive process by mathematical equation. It does not explicitly show the process of the recursive mechanism. For example, the tracing of the factorial function in calculating F(4) is presented as:

$$F(4) = 4 \text{ X } F(3) = 4 \text{ X } 3 \text{ X } F(2) = \ldots\ldots\ldots\ldots = 24.$$

Stack Simulation. (Dale & Lily, 1991; Greer, 1987; Tenenbaum & Augenstein, 1986) This approach is the one traditionally used in many textbooks. Recursion is introduced in terms of computer architectures for execution of recursive programs. Call to functions or procedures are traced with explicit reference to the system stack mechanism that is actually used in the Pascal implementation of recursion. This model is the concrete representation of the system stack, and in some cases is explained to the detailed level of activation records and return addresses. Often the concepts of stacks and activation records are new to students learning recursion and the local variables, procedure invocations, and procedure arguments must be manipulated in constructing the system stack. It is argued that this approach may add unnecessary burden to the learning of recursion (Ford, 1984; Greer, 1987). This approach, in fact, is a process tracing model but with a more concrete base domain.

Mathematical Induction. (Aho & Ullman, 1992; Ford, 1984; Greer, 1987; Henderson & Romero, 1989) This approach introduces recursion in terms of the mathematical basis for its correctness; that is, proof by induction. There are many similarities between recursion and mathematical induction. In both techniques, a base case(s) must be established first. Then, an assumption regarding the correctness of the solution for a particular size of the problem is made (inductive hypothesis), and then an extension to the next larger size of the problem is made. The base case and inductive hypothesis of mathematical induction may be translated directly into the recursive definition for a problem. Formulating a recursive algorithm is equivalent to formulating a mathematical induction.

One flaw of this approach is that students may or may not be familiar with mathematical induction before learning recursion. Ford (1984) suggested that a brief introduction of the concept may be necessary before teaching recursion and that it would be more appropriate to "argue" the correctness of a recursive algorithm rather than to "formal prove" it in the introductory level. This approach emphasizes the development of recursion as an instance of mathematical induction, seeks to improve relational understanding and consequently transfer of knowledge. It ignores the implementation details of recursion and is obviously an abstract model.

Structure Template. (Greer, 1987; Pirolli, 1985/1986a, 1986b) This model provides novice programmers with samples of recursive programs and describes the recursive algorithm in terms of base case(s) and recursive case(s). Pirolli and Anderson pointed out that knowing the underlying structure and functionality of recursive programs facilitated the induction skill needed for programming recursion more than knowledge of how such functions get executed. They concluded that the crucial factor in learning recursion seems to be the knowledge that recursive functions can be characterized as having base cases ("terminating case" in their term) and recursive cases ("recursive cases and recursive relations"" in their term). To solve a recursive problem is similar to filling out the slots of base case(s) and recursive case(s) in a structural template. A basic recursive template may look like:

```
IF <base condition> THEN <base action>  (* Base Case *)
    ELSE <recursive action>           (* Recursive Case *)
```

However, some researchers (Bhuiyan, Greer, & McCalla, 1989; Greer, 1987) suspected that students might try to memorize the templates used in different situations without a much deeper understanding of recursion. They observed that students tend to try to fill template slots on a trial and error basis. This approach is considered an abstract model.

2.5.3 Related Research

This section will review mainly experimental research related to teaching recursion. There appear to be three major issues that have been investigated: mutually influence on learning recursion and iteration, learning recursion from examples, and comparing effects of different conceptual models.

Kessler and Anderson (1986) looked at the transfer of skill between writing iterative and recursive computer programs. Their experiments involved 32 novice programmers using SIMPLE, a LISP-like language. They found positive transfer from writing iterative functions to writing recursive functions, but not vice versa. A subsequent protocol study of students' problem solving process revealed that subjects had a poor mental model of recursion that they developed poor learning strategies which hindered their understanding of iteration. Anzai and Uesato (1982) found similar results. They hypothesized that learning iteration first helps with recursion because iteration provides a model for what a recursive call does. However, learning recursion first does not help with iteration because iteration is easily mastered, and the recursion construct is not understood well enough to serve as the basis for transfer. Wiedenbeck's (1989) study provided directional, but not significant ($p < .10$), support for the above two studies. She concluded that the wisely suggestions for educational practice are probably to teach iteration before recursion.

Wiedenbeck (1989) also studied the extent to which students trained only with recursive examples are able to transfer their knowledge to compute other similar recursive mathematical functions in abstract form. It turned out that subjects who were subsequently given abstractly stated problems performed somewhat worse than they had performed previously when given examples. However, they did perform far better than a control group trained only with an abstract description of recursion (without examples). Pirolli and Anderson (1985) conducted an analysis and simulation model of verbal protocols of three subjects learning to program recursive functions. They found that problem solving by analogy to work-out examples is frequent in initial attempts by novices to write recursive functions. Subjects rely heavily on examples to guide their solutions to novel and difficult problems. It seems that providing extensive examples can serve as models to facilitate students' learning recursion.

In comparing the effects of conceptual models, Greer (1987) examined the effects of three models: architecture-oriented (stack simulation), theory-oriented (mathematical induction), and task-performance-oriented (structure template) models to teach college students recursive programming in Pascal. Three groups of students were taught recursion by means of video-taped lectures developed for each of the three models. No significant difference in achievement in recursion was found among the three groups. Students identified as having higher general computer science ability did demonstrate higher achievement in recursion, regardless of the conceptual models provided in instruction. Pirolli (1985/1986a) found subjects receiving the structure template model learned to program their recursive functions in less time than did subjects receiving the process tracing model. The performance (achievement) on the tasks between these two groups were not compared.

2.5.4 Implications for Instruction

From the previous discussion, the implications for teaching recursion can be summarized as follows:

1. Students' difficulty in learning recursion is mainly due to their inability to develop adequate mental models of the tasks. The provided conceptual models in instruction may play a crucial role here.
2. It is necessary to teach students the two basic skills: knowledge of how recursion works and knowledge of how to construct recursive algorithms. This implies that both glass box and black box approaches are important in teaching recursion.
3. Learning iteration first may help students when learning recursion, but not vice versa. (In fact, iteration is generally introduced before recursion in teaching a block-structure language such as Pascal.)
4. Providing extensive examples will facilitate novice students' learning recursion.

These implications serves as guidelines in developing the instructional framework for teaching recursion in the present investigation. According to the framework, the two sets of instruction materials (abstract and concrete) were designed, and their relative concreteness-abstractness were emphasized. The details of the instructional materials will be addressed in the following chapter.

2.6 SUMMARY

Norman (1983) stated that "As teachers, it is our duty to develop conceptual models that will aid the learner to develop adequate and appropriate mental models" (p. 14). A conceptual model of a system serves as an advance organizer by providing novices with a basic knowledge structure that can be used to build an initial mental model. Previous research has shown the effectiveness of conceptual models in teaching computing systems. Furthermore, in general, the conceptual models are most effective for domains that are novel or complex. Conceptual models can be categorized into two types of models -- abstract and concrete, according to the relative concreteness of their base domain and the detail level of internal process shown in the models. Literature does not provide conclusive findings for which type of conceptual model is better than the other.

Mayer is one of the few researchers who has studied the effects of conceptual models in computer programming domain. He repeatedly demonstrated the effectiveness of concrete models in teaching programming. However, Mayer's series research did not explore more complex conceptual knowledge such as the concept of data structures or the concept of flow of control (e.g., recursion or iteration). Neither did he investigate the effects of abstract models.

Researchers in mental models have pointed out that individuals' cognitive learning styles is one of the main individual difference features that affected the formation and acquisition of mental models. According to Kolb's learning styles theory, abstract learners take an analytical approach in learning and rely on systematic planning and logical thinking to solve problems; concrete learners take an experiential-based approach in learning and tend to rely more on their feeling and experiences in solving problems. Previous works have shown that abstract learners are likely to perform better than concrete learners. Bostrom et al. (1987) and Sein and Bostrom (1989) proposed that abstract learners may benefit more from an abstract model and concrete learners may benefit more from a concrete model. However, their studies did not provide decisive findings.

The reason for novices' difficulty in understanding recursion in programming is their inability to develop adequate mental models. This is because recursion is novel for novice programmers and recursive algorithms involve several different aspects of programming knowledge, which complicate the learning. Conceptual models appear to be appropriate tools for instruction in this kind of domain. Many conceptual models have been used in teaching recursion, but little research has been conducted to investigate their effectiveness. The major findings from the research literature are that prior learning of iteration and giving extensive examples may facilitate novice students' learning recursion.

Chapter 3 Research Method

3.1 Introduction

Recursion is a fundamental concept in computer science. Most computer science students have difficulty in learning recursion when the concept is first introduced. Conceptual models have been used in teaching recursion to help students understand this abstract concept. Researchers have discovered that students' learning styles may affect their learning and that there may be connections between the conceptual models provided and individuals' learning styles. The purpose of this study is to understand how conceptual models and learning styles influence novice programmers in learning recursion.

To fulfill the purpose of this study, an experimental research design was planned and implemented with a freshman computer science class at a major southwest research university. The following sections describe the entire research method.

3.2 Research Hypotheses

The research hypotheses tested in this experimental research study were:

H1: Students instructed in recursion with concrete conceptual models will outperform those instructed with abstract conceptual models on the *posttest* measure.

H2: Students instructed in recursion with concrete conceptual models will outperform those instructed with abstract conceptual models on the *retention* measure.

H3: Abstract learners will outperform concrete learners on the *posttest* measure.

H4: Abstract learners will outperform concrete learners on the *retention* measure.

H5: Abstract learners perform better on the *posttest* measure when provided with abstract conceptual models as opposed to concrete conceptual models.

H6: Abstract learners perform better on the *retention* measure when provided with abstract conceptual models as opposed to concrete conceptual models.

H7: Concrete learners perform better on the *posttest* measure when provided with concrete conceptual models as opposed to abstract conceptual models.

H8: Concrete learners perform better on the *retention* measure when provided with concrete conceptual models as opposed to abstract conceptual models.

3.3 Sample

The subjects for this study were students who enrolled in CS 304P at the major southwest research university in the Fall of 1992. CS 304P, Computer Science I, is the first required course for students who plan to major in computer science. It is also recommended for nonmajors who plan to take a second computer science course. The prerequisite for CS 304P is a score of at least 460 on the College Board Achievement Test in Mathematics at Level I, or three semester hours of mathematics with a grade of at least C. The major objective of this course is to introduce basic computer science concepts through programming with Pascal. Students enrolled in this course are considered novice programmers and have very little prior knowledge of computers in general or recursion in particular. This course requires students to attend a one-hour lecture section in addition to a two-hour discussion section each week. This course is a partially self-paced course, once students get two weeks ahead in the class they no longer have to attend the discussion sections. The lecture is taught by a course instructor; and the discussions are led by Teaching Assistants (TA) . There were a total of 12 discussion sections led by six TAs at the semester while this investigation was conducted.

The two experimental groups in this study were the *abstract model group* in which abstract conceptual models were used in teaching recursion, and the *concrete model group* in which concrete conceptual models were used. Students were assigned to the two groups by having each TA lead a section of both groups.

Each group, of approximately equal size, consisted of six discussion sections and were led by the same TAs. This avoided the possible variance caused by the TA factor.

The learning style of each student was identified through his/her results on the scrambled Kolb's Learning Style Inventory 1985 (LSI-1985, see section 3.6 Instrumentation). One major concern of the sampling process was that the distribution of students' learning styles might be unbalanced. Because the learning style of a student was determined by the norm provided in the LSI-1985 manual, it was possible to have 90% abstract learners and only 10% concrete learners. This would result in an inadequate number of subjects for the concrete learners' group. To try to ensure that enough subjects would be in both learning styles groups, a pilot test of students' learning styles was conducted on CS 304P students in the Spring of 1992. It was found that the distributions of abstract learners and concrete learners were about 60% and 40% respectively. Traditionally, more than 200 students enroll in CS 304P. This suggested that there would be enough subjects in each learning styles group for statistical analysis.

3.4 Experimental Design

The design of this experimental study is a pretest-posttest, 2 X 2 (conceptual models X learning styles) factorial design. The effects due to the conceptual models, learning styles, and the interaction of both in regards to recursion performance can be easily examined through this design.

Subjects were randomly assigned to either the abstract model group or the concrete model group based on which discussion section they attended. The treatment in the study was the different conceptual models used to present recursion to these two model groups. Within each model group, subjects were identified as either an abstract learner or a concrete learner based on their scores on the scrambled LSI-1985. Hence four treatment groups were formed: abstract learners with abstract models, abstract learners with concrete models, concrete learners with abstract models, and concrete learners with concrete models. To compare students' performance in the different groups, a posttest and two retention tests were conducted after the experimental treatment. There is evidence that students' prior knowledge in computer programming may affect their learning a new programming concept. Therefore a pretest was used to equate the variance caused by students' prior knowledge in conducting the statistical analysis.

The **independent variables** for this research were: (1) the conceptual models provided in teaching recursion, and (2) the student's learning style. The first independent variable consisted of two levels: abstract conceptual models and concrete conceptual models. The difference between the two models was that the abstract model emphasized the mathematical concept of recursion while the concrete model featured a more concrete demonstration of recursion. The second independent variable also had two levels: abstract learning styles and concrete learning styles. Students with an abstract learning style prefer logical thinking and develop theories to solve problems. Students with a concrete learning style rely on feeling and may learn better when provided with concrete examples.

The **dependent variables** in this study were the subjects' performance in recursive tasks on both the *posttest* and the *retention* tests. The posttest was intended to measure how much the subjects had learned about recursion right after the treatment. The retention tests, on the other hand, investigated the knowledge retained a period of time after the treatment was given. Two retention tests, one two weeks and the other six weeks after the treatment, were administered to the subjects.

3.5 EXPERIMENTAL TREATMENTS AND PROCEDURES

At the beginning of the Fall semester of 1992, the investigator met with the CS 304P instructor to arrange the appropriate time and procedures for the experiment. It was decided that the best way to do the experiment was to integrate the experimental procedures with the regular discussion section. This reduced students' extra work for participating in the study and resulted in more students in the experiment. Furthermore, to encourage students to participate, ten bonus points (out of 1000 points for the course) were given to students who completed the necessary procedures: signed a consent form, filled out the scrambled LSI-1985, attended the lecture on recursion, and completed the posttest.

The treatment in this study was the use of different conceptual models to teach recursion. A lecture based on the concrete conceptual model was presented to one group (concrete model group); and a lecture based on the abstract conceptual model was given to the other group (abstract model group). (See Instructional Materials of the Instrumentation section for both materials.)

One discussion section of 50 minutes duration was scheduled at the middle of the semester for the topic of recursion. The section was selected as the time for the treatment and the posttest. Iteration (which may facilitate the learning of recursion) and functions (which is a prerequisite for learning recursive functions) were taught at least a week before this discussion section. The course consisted of three major exams: review exam I, review exam II, and the final exam. Review exam I was held about two weeks before the lecture on recursion and was used as the pretest for the study. Review exam II and the final exam were scheduled two weeks and six weeks after the lecture on recursion, respectively; the retention tests were administered as part of both exams. Figure 3.1 shows the procedures for the experiment.

Two weeks before the treatment, the investigator attended the weekly TA meeting of the course and gave a brief introduction of the study and scheduled the time that recursion would be taught in each TA's discussion sections. The TAs were asked to explain the study to students, to collect the signed consent forms (Appendix B), and to administer the LSI instrument in their discussion sections the following week (Phase 1). Students who signed the consent form gave the investigator permission to access their exam scores in the course.

During the treatment week, the investigator visited all 12 discussion sections to present the lectures on recursion (Phase 2). During this time the TAs were not required to be in the classroom for the presentation. Six sections were given lectures using the abstract conceptual model and six sections were given lectures using the concrete model. The lecture (treatment) took about 35 minutes, and was followed by a 15 minute posttest. Students who were not participating in the study were also strongly encouraged to come to the presentation because recursion is a part of the course content and would be included in their review and final exams.

Phase 0 Pretest (two weeks before the treatment)

a. Subjects take the pretest. (review exam I of the course)

b. The investigator meets with the TAs and schedules the experiment.

Phase 1 Consent Form and LSI (a week before the treatment)

a. Subjects sign a consent form.

b. Subjects fill out a LSI. (10 minutes)

c. Subjects are assigned to two model groups.

Phase 2 Treatment and Posttest (treatment week)

a. The investigator lectures to each group. (35 minutes)

b. Subjects take the posttest. (15 minutes)

Phase 3 Retention Test 1 (two weeks after the treatment)

a. Subjects take the first retention test.
(embedded in review exam II of the course)

Phase 4 Retention Test 2 (six weeks after the treatment)

a. Subjects take the second retention test.
(embedded in final exam of the course)

Figure 3.1 Experimental Procedures

The first retention test (Phase 3) was embedded in review exam II. In order to avoid the possible variance from the TAs' comments on recursion, the TAs were asked not to discuss recursion in their discussion sections or during office hours before review exam II. If students had any questions regarding recursion, the TAs were to direct them to the investigator. The investigator held office hours between the period of the treatment and review exam II. The investigator answered students' questions using the appropriate treatment model. The second retention test (Phase 4) was embedded in the final exam. After review exam II, the TAs were free to discuss recursion with their students.

3.6 INSTRUMENTATION

Three kinds of instrument were used in this investigation: (1) a scrambled Kolb's Learning-Style Inventory 1985 (LSI-1985), (2) two sets of instructional materials, and (3) a pretest and three recursion achievement tests. Each of these will be described in greater detail below.

3.6.1 Learning-Style Inventory 1985 (LSI-1985)

The revised Kolb's LSI-1985 is an improved version of the original Learning-Style inventory developed by David A. Kolb in 1976. Like its predecessor, the LSI-1985 is designed to help individuals assess their ability to learn from experience. It measures an individual's relative emphasis on four learning orientations -- Concrete Experience (CE), Reflective Observation (RO), Abstract Conceptualization (AC), and Active Experience (AE) -- and computes two combination scores that indicate the extent to which the individual prefers abstractness over concreteness (AC-CE) and to which he/she emphasizes action over reflection (AE-RO). This study is interested in the abstract-concrete dimension of an individual's learning style. Therefore the AC-CE scale is the measure used in the study.

The LSI-1985 consists of 12 sentence-completion items in which respondents attempt to describe their learning styles. Respondents are required to rank-order (from 1 to 4) their preferences on four sentence endings that correspond to the four learning orientations described in Kolb's theory. It takes about ten minutes to complete the inventory. The resulting raw scores for the four basic scales, CE, RO, AC, and AE range from 12 to 48, and range from +36 to -36 for the two combination scales AC-CE (AC minus CE) and AE-RO (AE minus RO). To determine an individual's learning orientation along the abstract-concrete dimension, a norm is provided in the manual. If the AC-CE score is greater than or equal to four, the individual is classified as an abstract learner; otherwise the individual is classified as a concrete learner (Smith & Kolb, 1986, P. 101). The determination of the active-reflective dimension (AE-RO) uses a similar method.

Reliability

The internal consistency reliability for the four basic scales and two combination scores, as reported in the manual, range from .73 to .88, as measured by Cronbach's Standardized Scale Alpha (Smith & Kolb, 1986). Table 3.1 shows the internal consistency of the LSI reported in the literature and a pilot study conducted by the investigator. The mean coefficient alphas of the LSI-1985 for the four scales which range from .80 to .83 are considered high. The reliability of the two combination scales are not reported in the table because combined scales violate the assumption of independence for a coefficient alpha.

Table 3.1 Internal Consistency for the LSI

Study	N	CE	RO	AC	AE
LSI-1985					
Smith & Kolb (1986)	268	.82	.73	.83	.78
Sims et al. (1989)	317	.82	.84	.84	.86
Ruble & Stout (1990)	312	.85	.80	.83	.81
Pinto & Geiger (1991)*	55	.78	.81	.84	.86
Ruble & Stout (1991)	229	.82	.79	.81	.82
	Mean	.81	.80	.83	.83
Scrambled LSI-1985					
Ruble & Stout (1990)	323	.72	.72	.75	.73
Ruble & Stout (1991)	403	.67	.78	.78	.78
Veres et al. (1991)*	711/1042	.62	.67	.73	.55
Wu**	440	.77	.75	.81	.72
	Mean	.70	.73	.77	.70

* The coefficient alphas are the mean of two measures.
** See 3.9.1 Pilot Study 1.

Stability is another important property for a reliable measure. Smith and Kolb (1986) provide no test-retest reliability data for the LSI-1985 in the manual nor in subsequent articles. Table 3.2 gives the test-retest correlations for the LSI in several studies done by others. The test-retest reliability for LSI-1985 are moderate for all scales, except for the CE attribute.

Table 3.2 Test-retest Correlations for the LSI

Study	Interval	N	CE	RO	AC	AE	AC-CE	AE-RO
LSI-1985								
Atkinson (1988)	9 days	26	.57	.40	.54	.59	.69	.24
Atkinson (1989)	30 days	107	.49	.72	.67	.63	.59	.71
Pinto & Geiger(1991)*	1 yr	55	.25	.53	.59	.66	-	-
Ruble & Stout (1991)	5 wks	139	.18	.36	.46	.47	.22	.54
	Mean		.37	.50	.57	.59	.50	.51
Scrambled LSI-1985								
Ruble & Stout (1991)	5 wks	253	.37	.61	.59	.58	.48	.60
Veres et al. (1991)*	8 wks	711	.96	.97	.97	.96	-	-
	Mean		.67	.79	.78	.77	.48	.60

- The correlation of the scale was not reported.

* The coefficients are the mean of six measures.

While the internal consistency of the LSI-1985 is high, the stability of the inventory seems slightly low. This may be because of the response-set bias caused by the format of the instrument. The sentence endings corresponding with the four scales are presented in the same order for each of the 12 items of the inventory. Thus, a tendency to respond in the same fashion across items will give a high internal consistency, but may sacrifice the stability. A scrambled version of the inventory was suggested by Ruble and Stout(1990, 1991) and Veres, Sims, and Locklear (1991) to further improve the stability of the inventory. The four sentence endings within each item were randomly scrambled but the order of the 12 items within the LSI-1985 were not changed. As they predicted, the internal consistency of the scales was slightly decreased but the test-retest stability was improved. Table 3.2 and 3.3 show the reliability results of both versions of the LSI.

The scrambled LSI-1985 seems more reliable than the original LSI-1985 in that it takes out the response-set bias in the inventory. The mean coefficients for both internal consistency and test-retest reliability of the scrambled LSI-1985 are acceptable. The LSI used in the present investigation is a scrambled version of the LSI-1985. The original LSI-1985 has all the sentence endings corresponding to CE in the first column, RO in the second column, AC in the third column, and AE in the last column. The scrambled version of the LSI-1985 used in this investigation is located in Appendix C; and the scrambled item format for the version is in Appendix D.

Validity

Numerous researchers (Ferrel, 1983; Katz, 1986; Marshall & Merritt, 1985; Wilson, 1986) have examined and found support for Kolb's two bipolar dimensions: Abstract Conceptualization versus Concrete Experience (AC-CE) and Active Experimentation versus Reflective Observation (AE-RO). Most of the factor analytic research was done on the original LSI. Not much research has been done on the LSI-1985, and the findings regarding the validity of the LSI-1985 are not conclusive. Cornwell, Manfredo, and Dunlap (1991) and Ruble and Stout (1990) performed factor analysis on the LSI-1985 and the scrambled LSI-1985 and found the inventory failed to support the bipolar dimensions proposed in Kolb's theory. However, Katz (1986) and a pilot test (See section 3.9) conducted by the investigator do support the bipolar dimensions in Kolb's theory.

Unfortunately, there is no perfect LSI at the time of this investigation. After careful comparisons and study, Kolb's LSI seems a reasonable choice over other learning style inventories. Karrer (1988) examined five existing LSIs and concluded that all LSIs had weaknesses but Kolb's LSI was considered better than others. In addition, the reliability and validity of the scrambled LSI-1985 were verified in the pilot study conducted by this investigator. Therefore, the scrambled LSI-1985 was selected as the instrument for measuring students' learning styles in this investigation.

3.6.2 Instructional Materials

Two sets of instructional materials for teaching recursion were developed based on the concrete and abstract conceptual models respectively. The major difference between these two sets of instructional materials are in the *introduction* section of the lecture and the *verification* step of each given example. For the concrete model group, Russian Dolls serves as a literal metaphor in introducing recursion, and block diagram tracing is used to verify recursive functions (See <u>Concrete Instructional Material</u> below). For the abstract model group, recursion is introduced in terms of a recursive mathematical definition and verified by mathematical induction (See Abstract Instructional Material below).

The rationale and development of the two sets of instructional materials are described below.

Instructional Strategy

The instructional objectives of teaching recursion can be characterized by the following skills (Dale & Lilly, 1991; Dale & Weems, 1991; Greer, 1987). They are:

(1) to be able to read and understand recursive programs.

(2) to be able to write a recursive algorithm to solve a specific problem.

(3) to be able to evaluate when a recursive solution is appropriate for a given problem.

Objective (1) deals with *what is recursion* and *how does recursion work.* After knowing the definition and mechanism of recursion, objective (2) deals with *how to design a recursive solution* for a specific problem. Due to the limitation addressed in Chapter 1, the scope of recursive problems for this study is limited to recursive *functions* with *simple variables.* Problems involving recursive *procedures* or *structured variables,* which use the same concept but consist of more complicated language features, are not covered in this investigation. Objective (3), which seems too difficult for a novice programmer, is excluded in this investigation.

The strategy for teaching recursion is designed to achieve the first two objectives. To assure that the two sets of instructional materials have a parallel structure, a framework for teaching recursion is developed first. The implications for teaching recursion addressed in section 2.5.4. provide guidelines to design the framework. Next, the corresponding conceptual models for both sets of materials are put into the appropriate places to stress their *relative concreteness-abstractness*. The framework for teaching recursion, which is summarized in Figure 3.2, consists of the following six stages.

1. Introduction -- *What is recursion?*
2. Recursion in Pascal -- *How does recursion work?*
3. Designing recursive algorithms -- *How to design a recursive solution?*
   (a) Understand the problem
   (b) Determine the *size* of the problem
   (c) Identify the *base case(s)* of the problem
   (d) Identify the *recursive case(s)* of the problem
4. Verification -- *Verify* the implemented algorithm
5. Examples -- *Apply* the knowledge learned above to solve problems
6. Elaboration and Conclusion -- *Extend* the concept to solve more complicate problems

Figure 3.2 A Framework for Teaching Recursion

1. Introduction -- *What is recursion?* The definition of recursion is introduced. The base case(s) and recursive case(s) of a recursive algorithm are also defined in this stage. Different conceptual models may be given in the two sets of instructional materials. The introduction is a critical part in the teaching of a new concept. It not only bridges the new concept with prior knowledge (or experience) but also provides a firm background for learning the concept.

2. Recursion in Pascal -- *How does recursion work?* This stage transfers the abstract concept of recursion introduced above to application level, i.e., how recursion works in a Pascal program and how it really solves a problem. A recursive program is presented to students to explain the definition of recursion within the context of a programming language. The base case and recursive case of the program are identified and how recursion works is demonstrated by tracing the execution of a recursive function call. Different conceptual models can be used in tracing recursion for the two sets of materials.

3. Designing recursive algorithms -- *How to design a recursive solution?* This is one of the major goals for teaching recursion. Dale and Lilly (1991; pp. 474-475) provides a sound approach for designing a recursive solution:

(a) Get an exact definition of the problem to be solved.

(b) Determine the *size* of the problem to be solved. The size of the problem will decrease after each recursive call.

(c) Identify the *base case(s)* in which the problem can be expressed nonrecursively.

(d) Identify the *recursive case(s)* in which the problem is solved in terms of a smaller version of the same problem -- a recursive call.

4. Verification -- *Verify* the implemented algorithm by tracing the program (concrete model) or mathematical induction (abstract model).

5. Examples -- *Apply* the knowledge learned from the above to solve similar problems. This provides students an opportunity to summarize what they have learned in stages 1 through 4 and apply them to new problems. The software development method for solving programming problems (Koffman, 1992; pp. 13-14) is integrated here to solve each example problem:

(a) *Problem Specification.* Gain a clear understanding of what is required for the solution.

(b) *Analysis and Design.* Identify problem inputs, desired outputs, and any constraints for the problem. Design a recursive algorithm as stated in stage 3.

(c) *Implementation.* Implement the algorithm as a program, with Pascal.

(d) *Testing and Verification.* Test the completed program and verify that it work as expected using the verification method stated in stage 4.

6. Elaboration and Conclusion -- Extend the application of recursion to problems which require the use of Pascal procedures or structured variables, or which may have many base cases and/or recursive cases. The emphasis here is to tell students that there are other kinds of recursive problems, but the concept they just learned can be applied to all the problems. Because of time constraint, no example is given in this stage.

Three recursion problems: computing the *Factorial* of a number, *Summing* the total from 1 to N, and calculating the *Power* of a number, are presented in both sets of instructional materials. The Factorial problem is used in stages 2, 3, and 4, to address the concept of recursion and how to design and verify the recursive algorithm. The Summing and Power problems are discussed in stage 5 to give students an overview of the entire problem solving process. The difference between the two sets of instructional materials lies in the conceptual models applied in the presentations. The next two sections describe the two sets of instructional materials developed using the framework.

Concrete Instructional Material

The concrete instructional material set uses concrete conceptual models to present recursion. Stacked Russian Dolls, along with a block tracing diagram counting the dolls, is demonstrated in stage 1 to introduce the concept of recursion. The block tracing diagram physically mimics the attribute of Russian Dolls, where many successively smaller rectangles are drawn inside a large rectangle. The block tracing diagram is again used in stage 2 to show how recursion works and in stages 4 and 5 to trace the recursive algorithms developed. In identifying the recursive case(s) of a problem, "Find the relationship between the problem and a smaller version(s) of itself" is used as a hint to remind the students in the presentation.

The details of the materials are given in Appendix E and examples of the block tracing diagram are found in Appendix F. Fourteen transparencies (See Appendix G) were developed for presenting this approach.

Abstract Instructional Material

The abstract instructional material set employs abstract conceptual models in presenting recursion. Recursion is introduced using the recursive definition of a mathematical function. Then in stage 2, the mathematical equation operation (e.g., F(4) = 4 X F(3) = ...) is presented to show how recursion works. When verifying recursive algorithms, mathematical induction is applied to argue (but not formally prove) the correctness of the algorithm. Mathematical induction is briefly introduced in stage 4 by stating that the base/recursive case in recursion is similar to the base/inductive case in mathematical induction. The concept of mathematical induction is always used in determining the recursive case(s) of a problem: "*Assume a smaller version of the problem is true*, then find the relationship between the original problem and the smaller version of itself."

The material is found in Appendix H. Thirteen transparencies (See Appendix I) were developed for this approach.

Validating the Instructional Materials

Several revisions were done to improve the structure as well as the relative concreteness-abstractness of the materials.

The drafts of the materials were first reviewed by a local computer science education seminar whose members are computer science educators in the university. Then, the investigator lectured using both sets of materials and videotaped the presentations. A computer science instructor who has been teaching Pascal for years reviewed the tapes and gave opinions regarding the presentation, the transparencies used, and the materials presented. The results of this revision were used in the pilot study conducted in the summer of 1992. After the pilot study, the investigator decided to take out one example (computing the Nth Fibonacci number) from stage 5 because of the time constraint.

Two weeks prior to the treatment, the investigator presented the materials to the local computer science education seminar group. The last revisions were made and the final versions of both sets of materials were then produced.

3.6.3 Pretest and Recursion Achievement Tests

Pretest

The Pretest (See Appendix J) used in the investigation is a review test (i.e., review exam I) for CS 304P developed by the course instructor. The test consists of 35 multiple choice questions. It tests students' computer knowledge such as Boolean logic, assignment, If-Then-Else, and While loop. The results of this pretest were used as the covariate in the statistical analysis which would equate the variance caused by the students' prior knowledge. The internal consistency reliability of the pretest is .65, which is reasonably high for a non-standardized achievement test.

Three recursion achievement tests, a posttest and two retention tests, were developed for use in the investigation. The tests were designed to evaluate whether students achieved the course instructional goals relating to recursion. These goals are (1) to read and understand a recursive program and (2) to design a recursive solution for a specific problem. Therefore, two types of questions, predicting the results of a recursive program and generating the recursive definition (the base and recursive cases) of a problem, were included in the tests. Within each type of questions, two kinds of questions were given, one kind was similar to the examples given in the instruction (treatment) and the other kind was substantially different from the examples given in the instruction.

It is desirable to have open-ended questions rather than multiple choice questions in the tests to be used. Open-ended questions offer few or no clues for the answer and provide a wide variety or answers for analyzing students' conceptions, but they require more effort in grading. Multiple choice questions, on the other hand, are easier to grade but provide students with more opportunities to guess and may result in lower test reliability. The posttest, which was conducted by the investigator, was an open-ended format while the retention tests were multiple choice questions. This was because the retention tests were embedded in the CS 304P exams and needed to have the same format as the regular exams.

The questions in the recursion achievement tests were selected and revised from pilot tests conducted before this investigation. These pilot tests included: a quiz at the beginning of the follow-on course, and a pretest and three recursion achievement tests used in the pilot study administered through the summer of

1992. The item difficulty, appropriateness, and possible ambiguity of the questions were analyzed and used as a reference in designing the final achievement tests in this investigation.

Item difficulty is most relevant to achievement or aptitude tests and is usually defined statistically as the percentage of persons who respond correctly to an item. The higher the item difficulty value, the easier the item is considered to be, and vice versa. In general, it is better to have items with a range of item difficulties (e.g., ranging from .10 to .90) and with an average level of item difficulty around .50 (Walsh & Betz, 1985, pp. 69-72). This provides a wide distribution of the test scores and results in better discrimination among students' performance.

All the recursion achievement tests developed in this investigation were reviewed by the members in the local computer science education seminar group before they were given to the students. The content validity of the tests were satisfied through this process. The test scores collected in this investigation will be analyzed in order to determine the reliability and item difficulty data for each test. The three recursion achievement tests are described below.

Posttest

A copy of the posttest is located in Appendix K. This test consists of four questions. The ability to read and understand recursive programs is measured by question 1 and 3, which ask students to predict the results of recursive function/procedure calls. The ability to generate recursive algorithms for a problem is assessed through question 2 and 4, which require students to fill in the base case(s) and recursive case(s) for the problems. Question 1 and 2 are similar to examples given in the instruction; question 3 and 4 differ substantially from the examples. Question 1 is like the Summing problem presented in the instruction while question 2 has the same pattern as the Power problem. Question 3 deals with a *recursive procedure* and question 4 has *two base cases* as well as *two recursive calls* in the recursive case. Both types of questions were not discussed in the instruction because of the time limitation of the presentation.

The total score for the posttest is 16 points. Each question is worth four points, and thus two points for each blank slot except in question 2. In question 2, two points are given to the base case and the remaining two points are for the recursive case. Students were asked to show their work during the test. Partial credit was awarded if students did not get the correct answer but showed their understanding in their working out process. Partial credit was also given to semantically correct but syntactically wrong answers in questions 2 and 4.

The reliability for the posttest is .80 (N = 332), as measured by Cronbach's coefficient alpha. The data analyzed are the posttest scores collected in the study. The item difficulty for each question is given in Table 3.3. The difficulties range from .16 to .74 with an average difficulty .50. The test is considered to be a very reliable instrument based on the results described above.

Table 3.3 Item Difficulty for the Three Recursion Tests

Test	Q1	Q2	Q3	Q4	Q5	Mean
Posttest*	.72/.59	.74/.58	.50/.32	.35/.16	-	.50
Retention 1	.61	.74	.41	.78	.48	.60
Retention 2	.65	.77	.90	.57	.15	.61

* Every question has two subquestions.

Retention Test 1

Retention test 1 (See Appendix L) is a part of review exam II of CS 304P. This test contains five multiple choice questions. Questions 1 and 3 test predicting the result of a program and Questions 2, 4, and 5 test generating the recursive definition of a problem. Questions 1, 2, and 4 are similar to the examples given in the instruction while questions 3 and 5 are more elaborate questions. The total score is five points, one point for each question.

The reliability for the test is .53 (N = 453), as measured by Cronbach's Alpha. The low alpha value may be due in part to a poor reliability or to some other factors (e.g., test length or heterogeneous questions). Test length has an effect on the alpha value. In general, the more items there are on the test, the higher the alpha value for the test. The item difficulties for this test (Table 3.4) shows good distribution and a reasonable mean difficulty value. A further analysis of the test found the item discrimination for the five questions ranged from .28 to .48, which is greater than .20. This shows that the questions are well-designed. It can be concluded that the test is a reliable measure even though the alpha value is not very high.

Retention Test 2

Retention test 2 is embedded in the final exam of CS 304P. A copy of the test is located in Appendix M. This test consists of five multiple choice questions. Questions 1 and 2 test predicting the result of a program and Questions 3 - 5 test generating the recursive definition for a problem. Questions 1, 3, and 4 are similar to the examples given in the instruction while questions 2 and 5 are more complicated questions. The total score is five points, one point for each question. The reliability for the test is .62 (N = 400) which is satisfactory.

3.7 DATA COLLECTION

Scores on the *scrambled LSI-1985*, the *pretest* score, and three recursion achievement scores (i.e., the *posttest*, *retention test 1*, and *retention test 2*) were collected for each student in the experiment. The scrambled LSI-1985 test scores were collected by the TAs a week before the treatment. The pretest scores were provided by the instructor of CS 304P after review exam I. The posttest scores were gathered by the investigator immediately after the treatment. The two retention scores were extracted from the recursion questions in review exam II and the final exam in CS 304P. Students were asked to fill in their names and student ID numbers on all the tests in order to match their data in the experiment.

3.8 DATA ANALYSIS

Subjects who had studied recursion before the treatment were precluded from the data analysis of the investigation. The information was collected by asking subjects the following question: "Have you studied recursion before?" at the beginning part of the posttest.

A two-way Analysis of Covariance (ANCOVA) was performed on all data in order to test all the hypotheses. If F was significant at .05 level, then Scheffe's test was conducted to test the significance of the difference between the groups. In any investigation that involves learning, prior experience could influence how well subjects perform on a task. Hence, students' prior knowledge in programming (measured by the pretest) was the covariate in the analysis of performance. A Homogeneous Slopes test was run to test if the covariate satisfied the assumption of ANCOVA prior to the ANCOVA analyses.

The Cronbach's Standardized Scale Alpha was employed to further analyze the internal consistency reliability of the instruments used in the investigation. The statistical package SAS 5.18 in an IBM mainframe computer was used to perform all the data analysis.

3.9 Pilot Study Results

Two pilot studies were conducted prior to the present investigation. Pilot study 1 verifies the scrambled LSI used in the study. Pilot study 2 is similar to the current investigation but with a small sample. The instruments and research procedures were improved through the results of the pilot studies.

3.9.1 Pilot Study 1

The purpose of pilot study 1 was to determine the reliability and construct validity of the scrambled LSI-1985. Another objective of this pilot study was to investigate the distribution of students' learning styles in order to ensure that there would be enough subjects in the different learning styles groups for the present investigation. The scrambled LSI-1985 was administered to N = 440 undergraduate students at a major southwest research university during the spring of 1992. The subjects were enrolled in the following classes: Computer Science I (CS 304P), Computer Science II (CS 315), and Programming Languages (CS 345). The first two are lower division computer science classes while the last one is an upper division class.

The distribution of students' learning styles by class is shown in Table 3.4. Overall, 35% of the subjects were concrete learners and 65% were abstract learners. The percentage of the abstract learners increased as the level of the class progressed. This supported Kolb's claim that individual's learning styles may be oriented to a certain direction which relates to their learning or working environment. The distribution of CS 304P students were 41% concrete learners and 59% abstract learners. This suggested that there would be enough subjects available in both learning style groups for the purpose of this investigation.

Table 3.4 Distribution of Learning Styles by Class for Pilot Study 1

Learning Styles	CS 304P	CS 315	CS 345	Total
Concrete Learner	97 (41%)	44 (30%)	14 (27%)	155 (35%)
Abstract Learner	142 (59%)	105 (70%)	38 (73%)	285 (65%)
Total	239	149	52	440(100%)

Table 3.5 Distribution of Learning Styles by Sex for Pilot Study 1

Learning Styles	Female	Male	Total
Concrete Learner	46 (40%)	102 (34%)	148
Abstract Learner	70 (60%)	201 (66%)	271
Total	116 (28%)	303 (72%)	419 (100%)

* Twenty-one students did not fill out their sex identity.

Table 3.5 shows the distribution of students' learning styles by sex. The percentage of both learning styles in male and female students are about the same, which indicates that sex is not a factor in students' learning styles. Pinto and Geiger (1991) and Allinson and Hayes (1990) also found no significant differences due to sex on the learning style scales. Therefore, the sex factor was not considered in the present investigation.

The results of subjects' scores on the scrambled LSI-1985 were analyzed to examine the *reliability* and *validity* of the inventory. The reliability for the four basic scales all showed good internal consistency, which were .77 (CE), .75 (RO), .81 (AC), and .72 (AE), as measured by Cronbach's Standardized Scale Alpha (N = 440). According to Wiersma and Jurs (1990, p. 196), if an instrument consists of a number of subscales, the scores on the subscales can be factor analyzed to determine if they follow some hypothesized pattern. If they do, this would support the construct validity of the instrument with respect to the hypothesized pattern. A SAS program was employed to perform a principal factor analysis on the four scales. A two-factor solution, with varimax rotation was run to verify the construct validity of Kolb's two bipolar dimensions.

Table 3.6 Factor Loading for the Scrambled LSI (N = 440)

LSI Dimensions	Factor 1	Factor 2
CE	-.83	-.05
RO	-.03	.86
AC	.89	-.01
AE	-.06	-.81
Variance	.371	.347

The results of factor analysis is given in Table 3.6. The two bipolar dimensions are very clearly shown in the table. The Concrete Experience (CE) dimension is loaded negatively on the first factor, while the Abstract Conceptualization (AC) dimension loaded positively. As for the second factor, the Reflective Observation (RO) dimension is loaded positively whereas the Active Experimentation (AE) dimension loaded negatively. The two factors together accounted for 71.8% of the variance for the scrambled LSI-1985, which is reasonably high. The two factors were about equally important in explaining the variance, as the first factor accounts for 37.1%, and the second factor 34.7%. These results provided support for the scrambled LSI-1985 as a measure of the two bipolar dimensions proposed by Kolb's experimental learning theory.

3.9.2 Pilot Study 2

Pilot study 2 allowed the investigator to strengthen the present investigation in terms of the sampling process, experimental procedures, as well as the instruments. In particular, it enhanced the parallel structure of the two sets of materials and time control of the presentation (treatment) in the current investigation.

The pilot study was conducted during the summer of 1992. The subjects were students who enrolled in CS 304P at a major southwest research university. A total of 121 students enrolled in the course and 45 of them volunteered to attend and complete the experiment. There were four discussion sections led by two TAs. Each conceptual model group was compound of two discussion sections which were led by different TAs. The discussion section was 75 minutes long and scheduled twice a week. The time for the treatment was set at the discussion section that held one week after students were taught function construct.

A week prior to the treatment, students were asked to fill out the scrambled LSI-1985 and to sign the consent form. A pretest was given right before the treatment (10 minutes). The treatment was the presentation of recursion given by the investigator to each discussion section (approximate 40 minutes). A posttest which had been pilot tested at the beginning of the summer was administered immediately after the treatment (25 minutes). Two retention tests were embedded as a portion of the review exam II and the final exam of the course. The investigator held office hours for recursion questions before both exams. The TAs were asked not to discuss recursion with their students.

Results

The distribution of the subjects in each group is summarized in Table 3.7.

Table 3.7 Subjects Distribution of Pilot Study 2

	Abstract Model Group	Concrete Model Group	Total
Concrete Learners	8	5	13 (29%)
Abstract Learners	15	17	32 (71%)
Total	23	22	45 (100%)

There were 13 concrete learners, and only five of them in the concrete model group. The inadequate number of concrete learners in both model groups results in reduced statistical power in data analysis and thus any conclusion drawn from the results should be very cautiously.

The reliability of the achievement tests, as measured by Cronbach's Alpha, were: .67 (n = 85) for the pretest, .89 (n = 89) for the posttest, and .51 (n = 121) and .43 (n = 113) for the two retention tests. The reliability of the two retention tests did not seem high enough. This might be because both tests were short. The test length has an effect on the reliability alpha value. The posttest consisted of five programs and a total of 15 questions while the two retention tests consisted of five and three questions, respectively. Since the retention tests were embedded in the exams of the course, the number of recursion questions was restricted. Further item analysis might be necessary to evaluate the reliability of the two tests. The data for the two retention tests were not analyzed due to their undecided reliability.

The SAS ANCOVA procedure was employed to analyze the posttest data in pilot study 2. Table 3.8 presents cell data, and Table 3.9 presents the summary results of the ANCOVA run on the posttest scores with the pretest scores serving as the covariate. There were no main effects. No significant difference was found between different types of conceptual models, F(1,40) = 0.81, p = .37; and between different types of learning styles, F(1,40) = 2.81, p = .10, on the posttest measure. Nor were the interaction effects between conceptual models and learning styles detected, F(1,40) = 3.30, *p* = .08. Even though no statistical differences were found, some directional patterns can still be seen from the results. The subjects in the concrete model group (adjusted Mean = 13.3) seemed to perform better than the abstract model group (adjusted Mean = 11.6) while the abstract learners (adjusted Mean = 14.2) outperformed the concrete learners (adjusted Mean = 10.7). As for the interaction effects, the abstract learners seemed to benefit more from the abstract models (adjusted Mean = 15.1); whereas the concrete learners were hindered by the abstract models (adjusted Mean = 8.1).

Table 3.8 Descriptive Statistics on the Posttest Measure for Pilot Study 2

Group	N	Mean	SD
Abstract Learner, Abstract Model	15	15.2	4.5
Abstract Learner, Concrete Model	17	13.3	5.0
Concrete Learner, Abstract Model	8	7.9	5.4
Concrete Learner, Concrete Model	5	13.2	4.2

Table 3.9 ANCOVA Results on the Posttest Measure for Pilot Study 2

Source	SS	df	MS	F	*p*
Conceptual Models	26.79	1	26.79	0.81	.37
Learning Styles	92.92	1	92.92	2.81	.10
Interaction	109.27	1	109.27	3.30	.08
Error	1322.80	40	33.07		

$p < .05$

Conclusion

Many flaws were found in this pilot study upon careful review and study. These flaws might be responsible for the inconclusive findings. Appropriate procedures were implemented to avoid similar flaws in the present investigation. The following are the discussion of the flaws and how it was improved.

Sample Size. The sample size in the pilot study was too small, thus the results were less conclusive because of low statistical power. The 45 subjects accounted for only 37% of the students enrolled in the course in the summer of 1992. The subject recruiting procedures used in the current investigation were enhanced by asking the course instructor and the TAs to advocate the learning outcome of attending the experiment as well as giving additional credits for participating. The number of subjects were increased to 237, which accounted for 52% of the total enrolled students, because of the improved sampling procedures and the larger enrollment in the regular semester.

Presentation. The investigator presented recursion to the four discussion sections. It was found that the time spent in each section were not quite the same and even the materials presented to the same model group were not very consistent. To gain better control of the time and the materials presented, more transparencies were developed in order to cover more details of the presentation. All the lectures/presentations in the present investigation followed the materials shown in the transparencies.

Achievement Tests. The slightly low reliability alpha values of the two retention tests might be due to the fact that both tests were short. Further item analysis would be necessary in order to guarantee the quality of the questions. Test questions in the present investigation were adapted and/or revised from the tests in pilot study 2 with the consideration of their item difficulties and item discrimination.

No statistically significant effects were found in the ANCOVA analysis of the conceptual models and the learning styles in pilot study 2. But, the directional patterns for the two factors found in the pilot study were similar to several previous research findings. A careful research design as in the current investigation, which avoids the flaws in pilot study 2, is necessary to assure a more conclusive finding.

3.10 Summary

This chapter provided an overview of the research design of this investigation, including the sample selection, the experimental treatment and procedures, the development of the instruments, and the summary of two pilot study results. The results of the investigation will be presented in the next chapter.

Chapter 4 Research Findings

This investigation examined the effects of conceptual models and cognitive learning styles in novices learning recursion. This chapter contains an overview of statistical procedures, the results of data analyses, and a summary of the findings.

4.1 Overview of Analysis Procedures

Of the 453 students registered for an introductory computer science course (CS 304P) at a major southwest research university at the Fall of 1992, 280 (61.8%) students completed the pretest, signed the consent form, filled out the scrambled LSI-1985, and took the posttest. Students in one of the 12 discussion sections (ten students) were taught recursion before the treatment by their TA. Thirty-three students indicated they had studied recursion before. These two groups of students were excluded from the final data analysis. Therefore, N = 237 (52% of the enrolled students) was the sample size for this investigation. Table 4.1 reveals the distribution of subjects in each group. The number of subjects decreased slightly for retention analysis because a few students did not take the retention tests and a few dropped the course; N is 216 for retention test 1 and is 209 for retention test 2.

Table 4.1 Distribution of Subjects

	Abstract Model Group	Concrete Model Group	Total
Concrete Learners	37	39	76 (32%)
Abstract Learners	75	86	161 (68%)
Total	112	125	237 (100%)

A two-way Analysis of Covariance (ANCOVA) was performed in order to test all the hypotheses. If F was significant at .05 level ($p < .05$), then Scheffe's test was conducted to test the significance of the difference between the groups. Students' prior knowledge in programming (as measured by the pretest) was the covariate in the ANCOVA analyses. A Homogeneous Slopes test was run to test if the covariate satisfied the assumption of ANCOVA prior to the ANCOVA analyses. The statistical package SAS 5.18 on an IBM mainframe computer was used to perform all the analysis.

4.2 Results of Data Analysis

The results of the posttest and the two retention tests are analyzed in the following sections in order to test all the hypotheses.

4.2.1 Analysis of the Posttest

The posttest measured subjects' understanding of recursion after the treatment. It was designed to test Hypotheses 1, 3, 5, and 7. A Homogeneity Slope test was conducted prior to doing the ANCOVA calculation. The homogeneity test was not significant ($p = .57$), which means it was safe to proceed with the ANCOVA analysis. The descriptive statistics for the ANCOVA analysis are in Table 4.2. Table 4.3 presents a summary result of the ANCOVA analysis on the posttest measure.

Table 4.2 Descriptive Statistics on the Posttest Measure

		Pretest		Posttest	
Group	N (237)	Mean	SD	Mean	SD
Conceptual Models					
Concrete Model	125	26.5	3.5	7.9	3.9
Abstract Model	112	26.4	3.3	6.8	4.1
Learning Styles					
Concrete Learners	76	25.9	3.3	6.2	3.7
Abstract Learners	161	26.7	3.4	7.9	4.0
Learning Styles x Conceptual Models					
Abstract x Abstract	75	26.5	3.5	7.2	4.1
Abstract x Concrete	86	26.9	3.4	8.5	3.9
Concrete x Abstract	37	26.3	2.9	5.8	3.8
Concrete x Concrete	39	25.5	3.7	6.7	3.7

Table 4.3 ANCOVA Results on the Posttest Measure

Source	SS	df	MS	F	*p*
Conceptual Models	67.57	1	67.57	5.09	.03
Learning Styles	82.94	1	82.94	6.24	.01
Interaction	0.28	1	0.28	0.02	.89
Error	3082.81	232	13.29		

$p < .05$

The results of hypotheses testing are as follows:

H1: Students instructed in recursion with concrete conceptual models will outperform those instructed with abstract conceptual models on the *posttest* measure.

The conceptual models main effect was significant, F(1,232) = 5.09, p = .03. The concrete model group (adjusted Mean = 7.7) performed better than the abstract model group (adjusted Mean = 6.5) on the posttest measure. The results supported this hypothesis. Therefore, students instructed in recursion with concrete models outperformed those instructed with abstract models on the posttest measure, regardless of their learning styles.

H3: Abstract learners will outperform concrete learners on the *posttest* measure.

The learning styles main effect was also significant, F(1,232) = 6.24, *p* = .01. Abstract learners (adjusted Mean = 7.8) outperformed concrete learners (adjusted Mean = 6.5) in posttest performance, regardless of the conceptual models provided in instruction. The final results support the hypothesis.

H5: Abstract learners perform better on the ***posttest*** measure when provided with abstract conceptual models as opposed to concrete conceptual models.

H7: Concrete learners perform better on the ***posttest*** measure when provided with concrete conceptual models as opposed to abstract conceptual models.

Hypotheses 5 and 7 test the interaction effects between conceptual models and learning styles on the posttest measure. There was no interaction effect detected, F(1,232) = 0.02, *p* = .89. Therefore, these two hypotheses were not supported. Abstract learners did not benefit more from abstract models, and concrete learners did not benefit more from concrete models as measured in the posttest.

4.2.2 Analysis of the Retention Tests

The two retention tests were used to compare the effects of conceptual models and learning styles after a period of two and six weeks of the treatment. The retention tests were designed to test Hypotheses 2, 4, 6, and 8. A Homogeneity Slope test was conducted prior to doing the ANCOVA calculation on both retention measures. The homogeneity test was not significant for either retention test 1 ($p = .56$) and test 2 ($p = .21$). The descriptive statistics and the summary ANCOVA results on both retention measures are summarized in Table 4.4, 4.5, 4.6, and 4.7.

Table 4.4 Descriptive Statistics on Retention Test 1 Measure

		Pretest		Posttest	
Group	N (216)	Mean	SD	Mean	SD
Conceptual Models					
Concrete Model	119	26.6	3.5	3.0	1.3
Abstract Model	97	26.4	3.4	2.7	1.3
Learning Styles					
Concrete Learners	71	25.8	3.4	2.5	1.3
Abstract Learners	145	26.8	3.5	3.0	1.3
Learning Styles x Conceptual Models					
Abstract x Abstract	65	26.5	3.6	2.9	1.2
Abstract x Concrete	80	27.1	3.4	3.2	1.3
Concrete x Abstract	32	26.3	3.0	2.5	1.4
Concrete x Concrete	39	25.5	3.7	2.5	1.2

Table 4.5 ANCOVA Results on Retention Test 1 Measure

Source	SS	df	MS	F	*p*
Conceptual Models	1.32	1	1.32	0.89	.35
Learning Styles	7.88	1	7.88	5.34	.02
Interaction	0.27	1	0.27	0.18	.67
Error	311.55	211	1.48		

$p < .05$

Table 4.6 Descriptive Statistics on Retention Test 2 Measure

		Pretest		Posttest	
Group	N (209)	Mean	SD	Mean	SD
Conceptual Models					
Concrete Model	111	26.5	3.6	3.6	1.3
Abstract Model	98	26.3	3.4	3.5	1.4
Learning Styles					
Concrete Learners	69	25.9	3.4	3.3	1.4
Abstract Learners	140	26.6	3.5	3.7	1.3
Learning Styles x Conceptual Models					
Abstract x Abstract	65	26.3	3.6	3.7	1.3
Abstract x Concrete	75	27.0	3.4	3.7	1.3
Concrete x Abstract	36	26.4	3.0	3.1	1.6
Concrete x Concrete	36	25.3	3.8	3.4	1.2

Table 4.7 ANCOVA Results on Retention Test 2 Measure

Source	SS	df	MS	F	*p*
Conceptual Models	1.98	1	1.98	1.22	.27
Learning Styles	5.21	1	5.21	3.21	.05
Interaction	2.68	1	2.68	1.65	.20
Error	331.28	204	1.62		

$p < .05$

The results of hypotheses testing are as follows:

H2: Students instructed in recursion with concrete conceptual models will outperform those instructed with abstract conceptual models on the *retention* measure.

There was no conceptual models main effect found on either retention measure, F(1,211) = 0.89, $p = .35$ for retention test 1; and F(1,204) = 1.22, $p = .27$ for retention test 2. The concrete model group (adjusted Mean = 3.0 and 3.7) performed marginally better than did the abstract model group (adjusted Mean = 2.7 and 3.5) on both measures. This hypothesis was weakly supported; however, the difference was not significant at the .05 level.

H4: Abstract learners will outperform concrete learners on the *retention* measure.

The learning styles main effect was significant on both retention measures, F(1,211) = 5.34, $p = .02$ for retention test 1; and F(1,204) = 3.21, $p = .05$ for retention test 2. Abstract learners (adjusted Mean = 3.0 and 3.7) outperformed concrete learners (adjusted Mean = 2.5 and 3.3) on both retention tests , regardless of the conceptual models provided in instruction. The hypothesis was supported.

H6: Abstract learners perform better on the ***retention*** measure when provided with abstract conceptual models as opposed to concrete conceptual models.

H8: Concrete learners perform better on the ***retention*** measure when provided with concrete conceptual models as opposed to abstract conceptual models.

Hypotheses 6 and 8 concern the interaction effects between conceptual models and learning styles on the retention measure. The interaction effect was not significant on either retention measure, F(1,211) = 0.18, p = .67 for retention test 1; and F(1,204) = 1.65, p = .20 for retention test 2. These two hypotheses were not supported by either retention measure. Abstract learners did not benefit more from abstract models, and concrete learners did not benefit more from concrete models on the retention measure.

4.3 Summary of Findings

Table 4.8 is a summary of the hypotheses testing. The findings of this investigation are summarized below.

Table 4.8 Summary of Hypotheses Testing

	Hypotheses	Results
	Conceptual Models Effects	
H1	Concrete models superior to abstract models on the *posttest* measure	Support
H2	Concrete models superior to abstract models on the *retention* measure	Weak support
	Learning Styles Effects	
H3	Abstract learners superior to concrete learners on the *posttest* measure	Support
H4	Abstract learners superior to concrete learners on the *retention* measure	Support
	Interaction Effects	
H5	Abstract models better for abstract learners on the *posttest* measure	No support
H6	Abstract models better for abstract learners on the *retention* measure	No support
H7	Concrete models better for concrete learners on the *posttest* measure	No support
H8	Concrete models better for concrete learners on the *retention* measure	No support

Concrete conceptual models were better than abstract conceptual models in teaching recursion to novice programmers. However, the teaching effects weakened several weeks after classroom instruction. The finding is concluded from the results of Hypotheses 1 and 2 testing.

Novice programmers with abstract learning styles performed better than those with concrete learning styles when learning recursion. The data collected from the three recursion performance tests used in this investigation all supported this claim.

Finally, no interaction effect was found between the conceptual models provided in instruction and novice programmers' learning styles when learning recursion. Abstract learners did not necessarily benefit more from abstract conceptual models, and concrete learners did not necessarily benefit more from concrete conceptual models.

Chapter 5 Conclusion

This chapter includes a brief summary of the research problem and methodology used, a discussion of the results, and implications of the results. Finally, recommendations for future research are suggested.

5.1 Summary

Most computer science students have difficulty in learning recursion when the concept is first introduced. The reason may be because of a lack of everyday analogies and the complexity that recursive programming involves. A conceptual model used as an advance organizer is considered as a useful tool in helping students learning in a domain of this type. There is evidence that individual differences such as cognitive learning styles may affect students' learning. Furthermore, there may be connections between the conceptual models provided and individuals' learning styles. The purpose of this study was to better understand how different types of conceptual models and cognitive learning styles influence novice programmers when learning recursion.

The problem with which this study is concerned is *Which of two conceptual models (concrete or abstract) will best help novice programmers with different cognitive learning styles (concrete or abstract) to learn recursion?*

An experimental research design was planned and implemented to study this research problem. The design was a pretest-posttest, 2 X 2 (conceptual models X learning styles) factorial design. Two hundred thirty-seven students enrolled in an introductory computer science class at a major southwest research university served as the subjects for this study. Subjects were randomly assigned to either an abstract model group or a concrete model group and the groups were of approximately equal size. The treatment in the study was the different conceptual models (abstract or concrete) used to present recursion to the two model groups. Within each model group, subjects were identified as either an abstract learner or a concrete learner based on their scores on the scrambled LSI-1985. Conceptual models and learning styles were the two independent variables of this experimental design.

To compare students' performance in the different groups, a posttest and two retention tests were administered after the treatment. These three performance tests were the dependent variables of this design. A pretest administered prior to the treatment was used to equate the variance caused by students' prior knowledge in the statistical analysis. The statistical procedure two-way ANCOVA was employed to analyze all of the performance data. Two pilot studies were carried out to verify the instruments and to intensify the experimental design and procedures prior to the current investigation.

The results of this study were presented in Chapter 4 and will now be reviewed in the next section of this chapter.

5.2 Discussion of the Results

The four research questions proposed in the investigation are discussed below.

5.2.1 Conceptual Models

Research Question 1:

Are concrete conceptual models better than abstract conceptual models in helping students to learn recursion?

The results of Hypotheses 1 and 2 provided answers to this research question. **Concrete conceptual models were better than abstract conceptual models in helping novice programmers to learn recursion.** However, the effect was weak several weeks after the treatment (instruction). These findings are in accordance with Mayer's series studies (1981, 1982, 1985, 1987, 1988) which provided evidence for the effectiveness of using concrete models in teaching programming. Mayer believed that a concrete model allows novices to "see the works" and consequently helps them assimilate new information in a more coherent and useful way. On the other hand, an abstract model hides the internal details of a system from the users. Novices are likely to assume the system is just not understandable and thus they tend to memorize algorithms that "work", and are not able to develop a real understanding of the system.

The findings differ from two other studies (Greer, 1987; Pirolli, 1985/1986a) on the effects of conceptual models in teaching recursion for several reasons. Pirolli found subjects receiving the abstract model (structure template model) learned to program their recursive functions in *less time* than did subjects receiving the concrete model (process tracing model). However, he did not compare the *performance (achievement)* on the tasks between the two models as the present investigation did and the sample size of his study was very small (N = 19).

Greer (1987) did not find any significant difference between the concrete (architecture-oriented approach) and abstract (theory-oriented and task-performance-oriented approaches) models in teaching recursion in Pascal. The major differences between his study and the present investigation lie in the subjects and the scope of recursion involved in the experiments. The subjects of his study were not novice programmers but typically had at least a semester of programming experience and had been briefly introduced to the concept of recursion in their previous course. The subjects for this investigation were enrolling in their first computer science course and had not studied recursion before. Also, the scope of recursion involved in Greer's study was far more extensive. The current investigation involved only recursive functions with simple variables; the Greer's study also involved recursive procedures and recursion with structural variables such as arrays and pointers. It is ideal to investigate recursion from this global aspect, but care must be taken when measuring students' performance. Students' inability to solve a recursive problem may not be because they do not understand recursion but because they cannot handle the more complicated structures involved in the problem such as lists or trees.

The result of Hypothesis 2 showed that there were no significant differences between the two types of conceptual models on the retention measures. The concrete model group only performed marginally better than did the abstract model group for the retention tests, $p = .35$ and .27, respectively. These inconclusive results may be due to the fact that only 35 minutes of instruction (treatment) were given in the present investigation and this may well be too short to demonstrate the retention effects. Luiten et al. (1980) provided support for this explanation. In their meta-analysis of 135 studies, they found that the effect of advance organizers (conceptual models, termed in this study) increased with time; that is, when the instruction in the experiments extended to several days or weeks as compared to a few hours, the retention effect was stronger.

Halasz and Moran (1982) suggested that a concrete model (analogical model in his term) is effective for communicating complex concepts to novices when used as a literary metaphor whose function is simply to illustrate some salient points of the target system, but it is dangerous when used as reasoning about computer systems. The problem arises when a learner tries to extract more structure or relationships from an analogy than is warranted (du Boulay, 1986). They suggested that reasoning is much better done with an abstract model. Bennett (1984) and Sein et al. (1987) provided evidences for this assertion. They found that the abstract model group performed better than the concrete model group in complex tasks; the effect was reversed in simple tasks. However, Bennett's study, which was criticized for using a very contrived and artificial task, and considered to have very limited external validity (Newell & Card, 1985).

The difference between the present investigation and Sein et al.s' study are two fold. First, the target domains were different. They investigated a mail filing system while this investigation examined the domain of programming. Secondly, the concrete model used in their study might be too 'concrete' to infer the target system. They used a file cabinet as the concrete model of a filing system. Halasz and Moran (1982) have pointed out that a concrete model such as a filing cabinet is better used as a literary metaphor. When someone says that a file system is like a filing cabinet, it is simple to infer that the computer file system functions as a storage and retrieval system, but it is a complicated task to work out in detail how computers and filing cabinets are similar. While in the present investigation, the concrete models used were a concrete object (Russian Dolls) used as a literary metaphor and a block tracing diagram to demonstrate the mechanism of recursion.

The current investigation supports the use of concrete models in teaching novices programming, particularly in teaching novices recursion. The quality of concrete models is a critical issue in instruction. Effective concrete models must not only have a relative concrete base domain but also need to demonstrate an appropriate level of detail of the internal process of a system.

5.2.2 Cognitive Learning Styles

Research Question 2:

Do students with an abstract learning style outperform students with a concrete learning style in learning recursion?

The results of Hypotheses 3 and 4 showed that **students with abstract learning styles performed better than those with concrete learning styles in learning recursion.** The effect was independent of the type of conceptual models used in instruction. The finding is in agreement with previous work (e.g., Cavaiani, 1989; van Merrienboer, 1988; Bostrom et al., 1987; Sein & Bostrom, 1989, Zuboff, 1988) which found that abstract or analytical learners tend to perform better than concrete or non-analytic learners in computer related domains. The result is also consistent with Kolb's theory. Research on the Kolb's LSI has found a strong correspondence between individuals' learning styles and the careers people choose. Individuals found in the same careers tend to have similar learning styles. In other words, there is a 'fit' between individuals' learning styles and the requirements of their careers. People in the field of computer science are more likely to have an abstract learning style (Smith & Kolb, 1986). Learning computer science requires using logic and symbols, abstracting concepts, developing theories and models, and systematically analyzing problems. Clearly, abstract learners are much more comfortable with these kinds of learning situations. This may be the reason why abstract learners performed better than concrete learners in learning recursion.

5.2.3 Conceptual Models X Learning Styles

Research Question 3:

Do students with an abstract learning style learn recursion better when provided with abstract conceptual models?

Research Question 4:

Do students with a concrete learning style learn recursion better when provided with concrete conceptual models?

These two research questions deal with the interaction effects between conceptual models used in instruction and an individual's cognitive learning style. The test of Hypotheses 5 through 8 provided the answer for these two research questions. There were no interaction effects detected on all three recursion performance measures (i.e., the posttest and two retention tests). These results suggested that **abstract learners did not necessarily benefit more from abstract conceptual models, and concrete learners did not necessarily benefit more from concrete conceptual models in learning recursion.**

Two studies (Bostrom et al., 1987; Sein & Bostrom, 1989) in the literature examined the interaction effects between conceptual models and learning styles in learning about computer systems (e.g., mail filing system). They proposed that abstract learners who take an analytical approach to learning would have the abilities to discover the rules and structures inherent in an abstract model. Conversely concrete learners take an experience-based approach to learning and tend to rely heavily on prior relevant experience drawn from a concrete model.

Therefore, abstract learners should benefit more from an abstract model, and concrete learners should benefit more from a concrete model. However, their experiments did not provide strong support for their claim.

Table 5.1 Summary of Studies on Interaction Effects

Study	Target System	Measure	Results*
Bostrom et al (1987)			
Study #1	Financial	Accuracy	NS
	Planning System	Time	NS
		Comprehension	NS
Study #2	Mail System	Accuracy	NS
		Time	NS
		Comprehension	NS
Study #3	Mail System	Accuracy	S
		Comprehension	NS
Study #4	Lotus 1-2-3	Accuracy	NS
		Comprehension	NS
Sein & Bostrom (1989)	Mail System	Accuracy	S
		Comprehension	NS
Wu (the present study)	Recursion	Comprehension	NS

*$p < .05$, NS: Not Significant, S: Significant

Table 5.1 is a summary of interaction effects between conceptual models and learning styles on the abstract-concrete dimension in these two studies and the present investigation. The measure *Accuracy* refers to whether a particular learning task was completed correctly. It was measured by the number of experimental tasks performed correctly. *Time* is the amount of time taken to complete a task. *Comprehension* is the score obtained on the post-training quiz which tested subjects' knowledge about specific functions, features, and application of the target software. Two studies on the *accuracy* measure supported their claim; no significant interaction effects were found on the remaining measures. It seemed, in general, that the interaction effects were very weak.

The inconclusive findings might be because of the different nature of the measures and the different target domains. The two interaction effects found in Table 5.1 were both in the *accuracy* measure of learning a *mail system*. The accuracy measure was measured through subjects' interacting with a computer system. A possible explanation is that the interaction effects were more likely to happen when students directly interact with a computer system, but not for using pens and papers (such as comprehension measure), to solve a problem. If this is the case, the match of conceptual models and learning styles may be helpful for students' debugging in programming, which requires students interacting with computers, but it does not provide help for comprehending an abstract concept such as recursion.

5.3 IMPLICATIONS

The importance of conceptual models in teaching/training a complex domain has been established in the literature. However, the form of conceptual models (concrete or abstract) that are more effective in helping novices learning has remained inconclusive. The same issue exists in using conceptual models in teaching recursion. The findings of this investigation suggest that concrete models are better than abstract models in teaching novice programmers recursion. Yet, for intermediate or experienced programmers who may or may not have prior knowledge about recursion, which type of conceptual models are favorable is still an open issue. Teachers should be very cautious in adapting or designing concrete models. A concrete model needs not only to employ an analogy from a relatively concrete (and familiar, if applicable) object but also to demonstrate the appropriate level of details of the internal mechanism as defined in this investigation. Several current introductory computer science textbooks (e.g., Dale & Weems, 1991; Koffman, 1992) have provided good examples of using concrete models to present recursion.

Though individual's learning style is not a measure of ability such as intelligence, some styles may be more effective than others in certain situations. Previous studies have shown that an individual's learning style can be a predictor of his/her success in certain career fields. The results of this investigation as well as other studies suggest that individuals with an abstract learning style tend to perform better in computer related fields. In other words, individuals with a concrete learning style will have more difficulty in learning computer science concepts such as recursion. As a teacher, it is important to identify these students and provide them with care and support necessary for success. Awareness of students' learning styles may assist teachers in aligning their teaching methods with their students, rather than to their own personal learning style. In addition, teachers can assist students in evaluating and adapting their own learning styles and can use more versatile learning strategies to adjust to the instructional setting (Davidson, 1990).

Theoretically, the matching of students' learning styles and appropriate conceptual models in learning a system or concept is a sound instructional approach. However, previous studies did not provide plausible findings. Neither did this investigation find any relationship between students' learning styles and conceptual models used in teaching recursion. A careful examination of Bostrom and his colleagues' studies (Bostrom et al., 1987; Sein & Bostrom, 1989) revealed that the relationship might exist in tasks which emphasized interacting with computers such as using a mail system. As for learning recursion, or programming in general, the matching of learning styles and conceptual models might be useful in tasks such as debugging programs.

5.4 Recommendations for Future Research

One limitation of the present investigation was the time of the treatment (instruction) which was limited to 35 minutes. This restriction may have resulted in the lack of retention effects seen between the conceptual models. The time limitation also restricted the coverage to only one aspect of recursion: recursive functions with simple variables. A replication study with a longer treatment period that covers more aspects of recursive programming is recommended. It is important to design a sensitive measure that can distinguish between the achievement in recursion and other aspects of programming (e.g., pointers) in the replication study.

The concern of this investigation was how conceptual models related to novice programmers learning recursion. Students' mental models of recursion were not analyzed in this study. Many studies have examined students' mental models of recursion (e.g., Kessler & Anderson, 1986; Pirolli, 1986a; Greer, 1987). However, they generally suffered from two methodological shortcomings: over-reliance on performance data and lack of ecological validity (one-shot study with a short time interaction between users and system) as described in Sasse's paper (1991). Bhuiyan et al. (1991) have done a preliminary study to explore students' mental models of recursion from the evolving aspects. More research which avoids the methodological shortcomings cited above should be conducted to better understand students' mental models of recursion.

The present study investigated the abstract-concrete (AC-CE) dimension of Kolb's learning styles in learning recursion. This dimension was considered more likely to have interaction effects with conceptual models used in instruction. It is also interesting to note how the other dimension, active-reflective (AE-RO), relates to students learning recursion. The active-reflective dimension deals with aspects of active involvement in learning and may have interaction effects with the type of instructional methods provided and is less related to the conceptual models used. For example, reflective learners would rely on observation and viewing things from different perspectives, but would not necessarily take any action in their learning. A traditional lecture-based instruction would be more appropriate. On the other hand, active learners prefer experimenting with changing situations and getting things done through action. An activity-based instruction such as group discussion or closed laboratory (described as in Tucker, 1991) would be more appropriate. Future research should investigate how the active-reflective dimension of learning styles relates to the instructional methods provided.

Finally, the match of learning styles with conceptual models is theoretically sound as discussed in Chapter 2. However the present investigation and previous studies provided few supports for this assertion. Research in this field is still too young to draw a definite conclusion. More research needs to be done in the field. As observed from the previous studies, the interaction effects seem more likely to exist in tasks (or domains) which require directly interacting with computer systems. Future research is also recommended to investigate the relationship between the characteristic of learning tasks (or domains) and the matching of learning styles with conceptual models.

Appendix A Introduction to Recursion

Recursion, *Recursive Function*, or *Recursive Procedure* is a mechanism for defining something in terms of a simpler version of itself. An example of recursion in mathematics is the factorial function:

$$f(n) = \begin{cases} 1, & \text{if } n = 0 \quad \textit{(Base Case)} \\ n \text{ X } f(n-1), & \text{if } n > 0 \quad \textit{(Recursive Case)} \end{cases}$$

in which, f(n) is defined in terms of f(n-1). The case (or cases) for which an answer is explicitly known is called the **base case**; the case for which the solution is expressed in terms of a simpler version of itself is called the **recursive** or **general case**. The computation of such a function is carried out by suspending the calculation of n X f(n-1) until f(n-1) is carried out, which in turn requires that (n-1) X f(n-2) be suspended until f(n-2) is carried out, and so on, until f(0) is reached. For instance, the value of f(3) is carried out as following.

$$\begin{aligned} f(3) &= 3 \text{ X } \underline{f(2)} \\ &= 3 \text{ X } \underline{2 \text{ X } \underline{f(1)}} \\ &= 3 \text{ X } 2 \text{ X } \underline{1 \text{ X } \underline{f(0)}} && (\, n = 0, \text{ Base Case }) \\ &= 3 \text{ X } 2 \text{ X } 1 \text{ X } \underline{1} && (\, f(0) = 1 \,) \\ &= 6 \end{aligned}$$

In a programming language, a function or procedure is called recursive if it calls itself. For example, a Pascal implementation of the factorial function would be as below:

```
FUNCTION f (n : Integer): Integer;
     BEGIN
     IF      n = 0
             THEN f := 1             (* Base Case *)
             ELSE f := n * f(n-1)    (* Recursive Case *)
     END;
```

Function f calls itself with parameter n-1 when n is not equal to 0. This recursive process will eventually stop when n reaches 0.

Appendix B Consent Form

Consent Form

You are invited to participate in a study of how student's learn recursion. We are studying the relationship between learning recursion and an individual's learning style. If you decide to participate, you will be asked to fill out a learning-style inventory which takes about ten minutes. This permission form allows us to obtain your examination scores in this course. Any information that is obtained remains confidential.

Your decision whether or not to participate does not prejudice your future relations with The University of Texas at Austin. If you decide to participate, you are free to discontinue participation at any time without prejudice.

If you have any questions, please ask us: Dr. Nell Dale, 471-7316, Instructor Suzy Gallagher, Dr. Lowell Bethel and Cheng-Chih Wu, 471-7354. We will be happy to answer your questions.

You may have a copy of this form to keep if you wish.

Your signature indicates that you have read the information provided above and have chosen to participate.

Signature	Date	ID number
Please Print Your Name Here	Signature of Investigator	

Appendix C Scrambled Learning-Style Inventory 1985

Learning-Style Inventory

Name ______________________ ID# ____________________

INSTRUCTIONS

On the following you will be asked to complete 12 sentences. Each has four endings. Rank the endings for each sentence according to how well you think each one fits with how *you* would go about learning something. Try to recall some recent situations where you had to learn something new, perhaps in your job or current classes. Then, using the spaces provided, rank a "4" for the sentence ending that describes how you learn best, down to a "1" for the sentence ending that seems least like the way you would learn.
Be sure to rank all the endings for each sentence unit. Please do not make ties.

Example of completed sentence set:

0. When I learn:	4 I am happy.	2 I am fast.	1 I am logical.	3 I am careful.

REMEBER: 4 = *most* like you
3 = *second* most like you
2 = *third* most like you
1 = *least* like you

AND: You are ranking across, not down.

1. When I learn:	__ I like to deal with my feelings.	__ I like to watch and listen.	__ I like to think about ideals.	__ I like to be doing things.
2. I am best when:	__ *I listen and watch carefully.*	__ *I rely on logical thinking.*	__ *I work hard to get things done.*	__ *I trust my hunches and feelings.*
3. When I am learning:	__ I tend to reason things out.	__ I am responsible about things.	__ I have strong feelings and reactions.	__ I am quiet and reserved.
4. I learn by:	__ *doing.*	__ *feeling.*	__ *watching.*	__ *thinking.*
5. When I learn:	__ I am open to new experiences.	__ I look at all sides of issues.	__ I like to analyze things, break them down into their parts.	__ I like to try things out.
6. When I am learning:	__ *I am an observing person.*	__ *I am a logical person.*	__ *I am an active person.*	__ *I am an intuitive person.*
7. I learn best from:	__ rational theories.	__ a chance to try out and practice.	__ personal relationships.	__ observation.
8. When I learn:	__ *I like to see results from my work.*	__ *I feel personally involved in things.*	__ *I take my time before acting.*	__ *I like ideas and theories.*
9. I learn best when:	__ I rely on my feelings.	__ I rely on my observations.	__ I rely on my ideas.	__ I can try things out for myself.
10. When I am learning:	__ *I am a reserved person.*	__ *I am a rational person.*	__ *I am a responsible person.*	__ *I am an accepting person.*
11. When I learn:	__ I evaluate things.	__ I like to be active.	__ I get involved.	__ I like to observe.
12. I learn best when:	__ *I am practical.*	__ *I am receptive and open-minded.*	__ *I am careful.*	__ *I analyze ideas.*

Appendix D Item Format for the Scrambled LSI-1985

Item	Column 1	Column 2	Column 3	Column 4
1	CE	RO	AC	AE
2	RO	AC	AE	CE
3	AC	AE	CE	RO
4	AE	CE	RO	AC
5	CE	RO	AC	AE
6	RO	AC	AE	CE
7	AC	AE	CE	RO
8	AE	CE	RO	AC
9	CE	RO	AC	AE
10	RO	AC	AE	CE
11	AC	AE	CE	RO
12	AE	CE	RO	AC

Appendix E Concrete Instructional Material

Objectives

1. To understand the definition of recursion
2. To read and understand recursive programs (Recognition)
3. To generate base cases and recursive cases of a recursive function (Generalization)

1. Introduction

A. Use ***Russian Dolls*** as the literal metaphor to convey the concept of recursion.

B. Define the Base Case and Recursive Case

2. Recursion in Pascal

Example 1 **Factorial Problem**

Write a recursive function to calculate n!.

A. Show the solved recursive program

B. Identify the base case and recursive case

C. Use ***Block Tracing Diagram*** to trace the result of 3!

3. Designing Recursive Algorithms

A. Understand the problem

B. Determine the *size* of the problem to be solved

C. Determine the base case(s)

D. Determine the recursive case(s)

4. Verification

Block Tracing Diagram to trace the solved program:

A. Trace base case

B. Trace a small size of recursive case

5. Examples

Example 2 **Summing Problem**

A. Problem Specification: Write a recursive function to calculate the result of $1 + 2 + 3 + \ldots + N$.

B. Analysis and Design: Determine the size, base case(s), and recursive case(s) of the problem.

C. Implementation: Complete the recursive program.

D. Verification: Trace both base case(s) and recursive case(s) using **Block Tracing Diagram.**

Example 3 **Power Problem**

A. Problem Specification: Write a recursive function to calculate a positive integer to a positive power. ($X^1 = X$; $X^N = X * X^{N-1}$)

B. Analysis and Design: Two parameters are required in this problem, which one is the size of the problem? Determine the base case(s) and recursive case(s).

C. Implementation: Complete the recursive program.

D. Verification: Trace both base case(s) and recursive case(s) using **Block Tracing Diagram.**

6. Elaboration and Conclusion

The mechanism of the recursion is the same in the following situations:

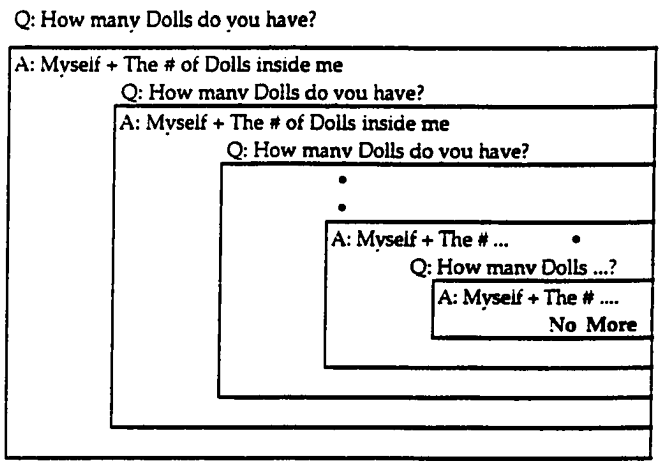
A. Many base cases and/or many recursive cases

B. Recursive Procedures

C. Structured Variables, e.g., arrays or linked list

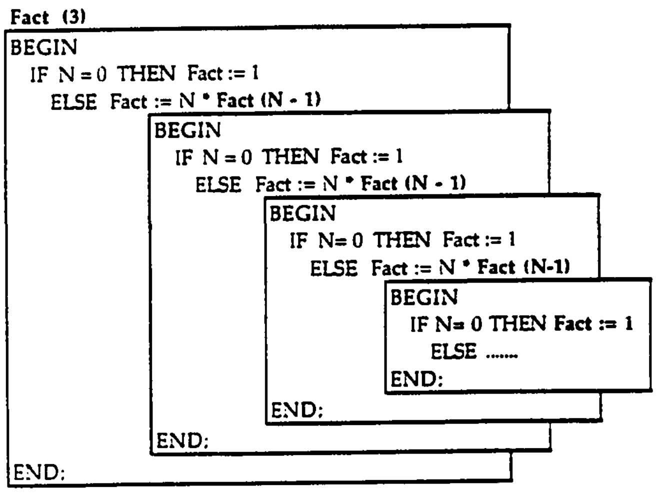
Appendix F Examples of Block Tracing Diagram

Russian Dolls



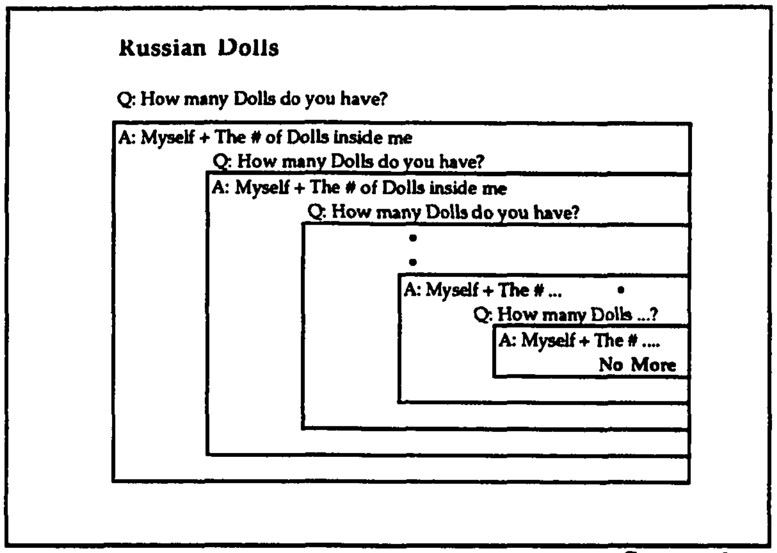
```
Q: How many Dolls do you have?
A: Myself + The # of Dolls inside me
        Q: How manv Dolls do vou have?
    A: Myself + The # of Dolls inside me
            Q: How manv Dolls do vou have?
                •
                •
            A: Myself + The # ...     •
                    Q: How manv Dolls ...?
                A: Myself + The # ....
                          No More
```

Tracing Fact(3)



```
Fact (3)
BEGIN
  IF N = 0 THEN Fact := 1
    ELSE Fact := N * Fact (N - 1)
    BEGIN
      IF N = 0 THEN Fact := 1
        ELSE Fact := N * Fact (N - 1)
        BEGIN
          IF N= 0 THEN Fact := 1
            ELSE Fact := N * Fact (N-1)
            BEGIN
              IF N= 0 THEN Fact := 1
                ELSE .......
            END;
        END;
    END;
END:
```

Appendix G Transparencies for Concrete Instructional Material



Russian Dolls

Q: How many Dolls do you have?

A: Myself + The # of Dolls inside me

Q: How many Dolls do you have?

A: Myself + The # of Dolls inside me

Q: How many Dolls do you have?

•

•

•

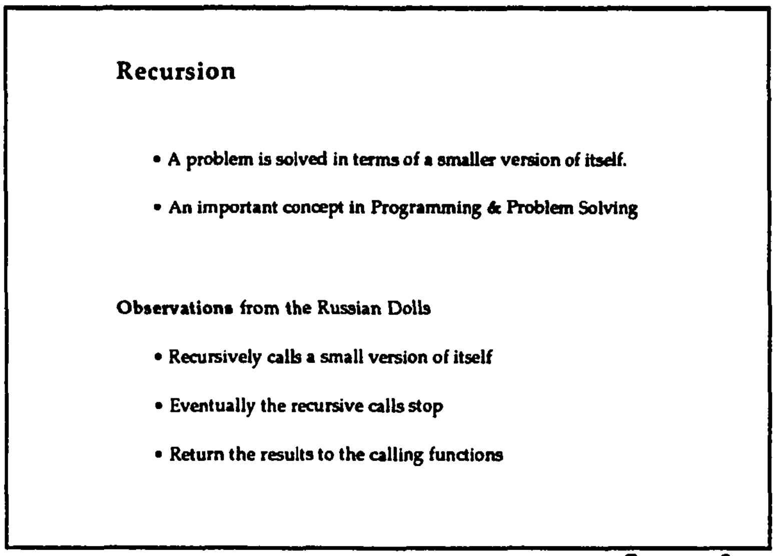
A: Myself + The # ...

Q: How many Dolls ...?

A: Myself + The # ....

No More

Concrete 1



Recursion

- A problem is solved in terms of a smaller version of itself.
- An important concept in Programming & Problem Solving

Observations from the Russian Dolls

- Recursively calls a small version of itself
- Eventually the recursive calls stop
- Return the results to the calling functions

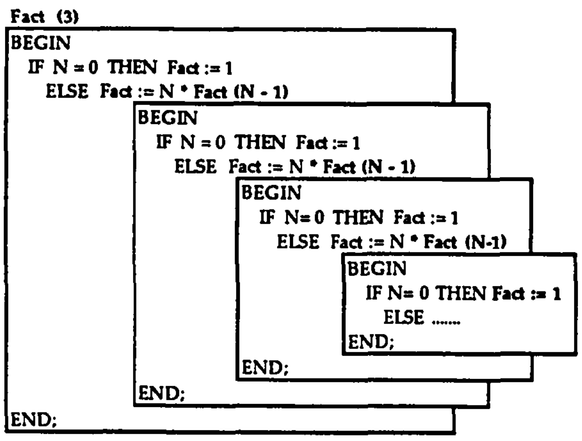
Concrete 2

Recursion in Pascal

```
PROGRAM RecuFunCall (Input, Output);
FUNCTION Fact (N: Integer): Integer;
        (* Compute the factorial of N ( i.e. N! ) *)
    BEGIN
        IF  N = 0   THEN   Fact := 1          (* Base Case *)
            ELSE   Fact := N * Fact (N - 1)   (* Recursive *)
    END;
BEGIN
    Writeln ( Fact (0) );          Writeln ( Fact (3) );
END;
```

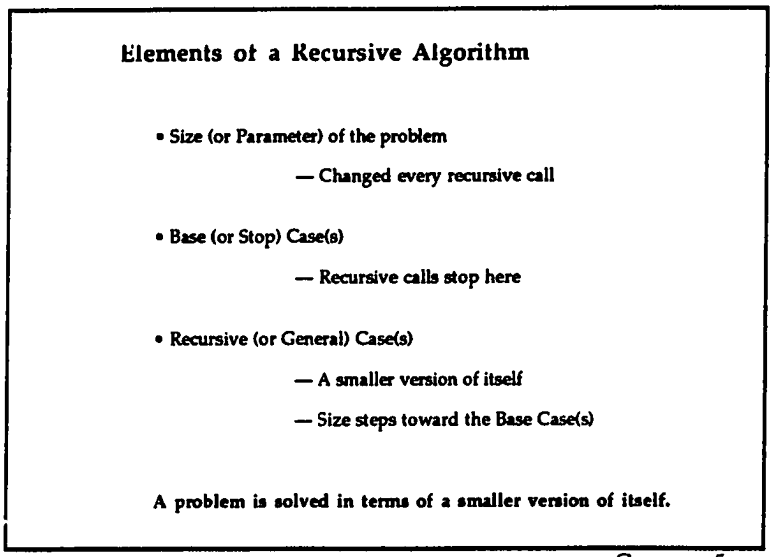
Concrete 3

Tracing Fact(3)



```
Fact (3)
BEGIN
  IF N = 0 THEN Fact := 1
    ELSE Fact := N * Fact (N - 1)
             BEGIN
               IF N = 0 THEN Fact := 1
                 ELSE Fact := N * Fact (N - 1)
                          BEGIN
                            IF N= 0 THEN Fact := 1
                              ELSE Fact := N * Fact (N-1)
                                       BEGIN
                                         IF N= 0 THEN Fact := 1
                                           ELSE .......
                                       END;
                          END;
             END;
END;
```

Concrete 4

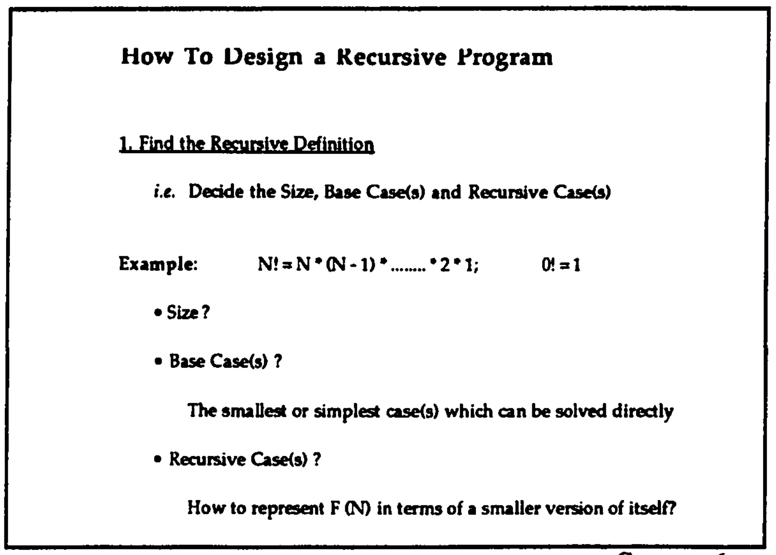


Elements of a Recursive Algorithm

- Size (or Parameter) of the problem
  - — Changed every recursive call
- Base (or Stop) Case(s)
  - — Recursive calls stop here
- Recursive (or General) Case(s)
  - — A smaller version of itself
  - — Size steps toward the Base Case(s)

A problem is solved in terms of a smaller version of itself.

Concrete 5



How To Design a Recursive Program

<u>1. Find the Recursive Definition</u>

i.e. Decide the Size, Base Case(s) and Recursive Case(s)

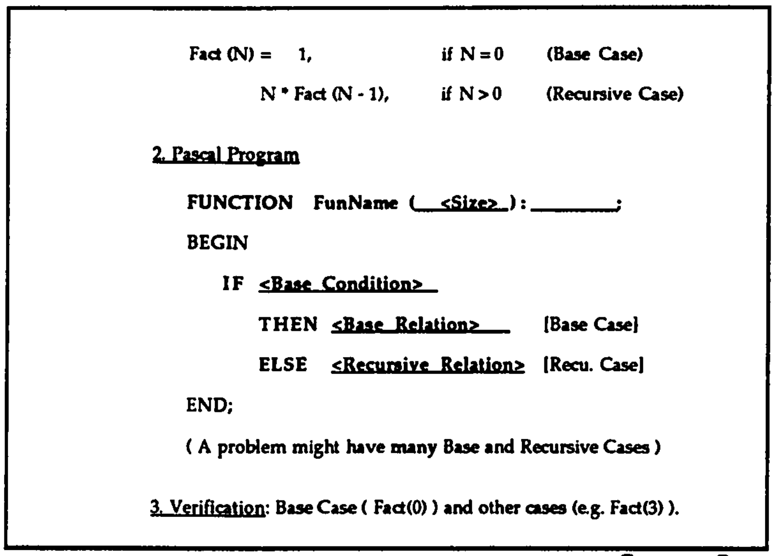
Example: $N! = N * (N-1) * \ldots\ldots * 2 * 1;$ $0! = 1$

- Size ?
- Base Case(s) ?

  The smallest or simplest case(s) which can be solved directly
- Recursive Case(s) ?

  How to represent F (N) in terms of a smaller version of itself?

Concrete 6



Fact (N) = 1, if N = 0 (Base Case)

N * Fact (N - 1), if N > 0 (Recursive Case)

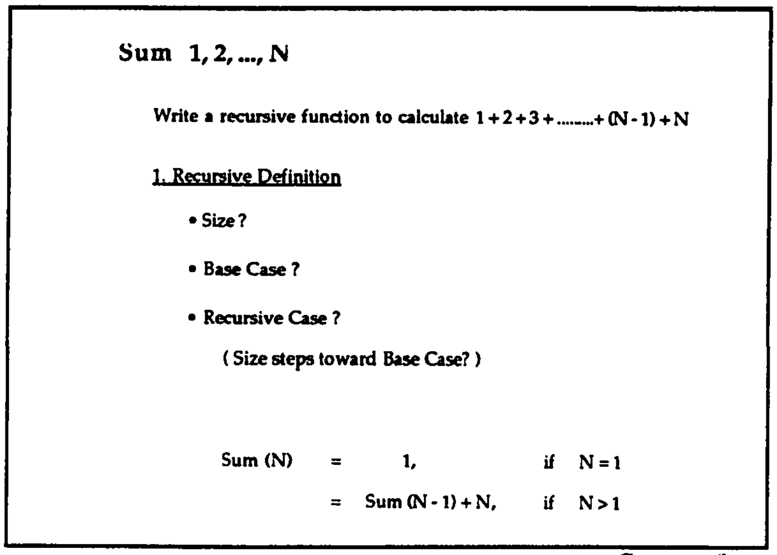
2. Pascal Program

```
FUNCTION  FunName ( <Size> ) : ________;
BEGIN
   IF <Base Condition>
      THEN <Base Relation>       [Base Case]
      ELSE <Recursive Relation>  [Recu. Case]
END;
```

(A problem might have many Base and Recursive Cases)

3. Verification: Base Case (Fact(0)) and other cases (e.g. Fact(3)).

Concrete 7



Sum 1, 2, ..., N

Write a recursive function to calculate 1 + 2 + 3 + ...+ (N - 1) + N

1. Recursive Definition

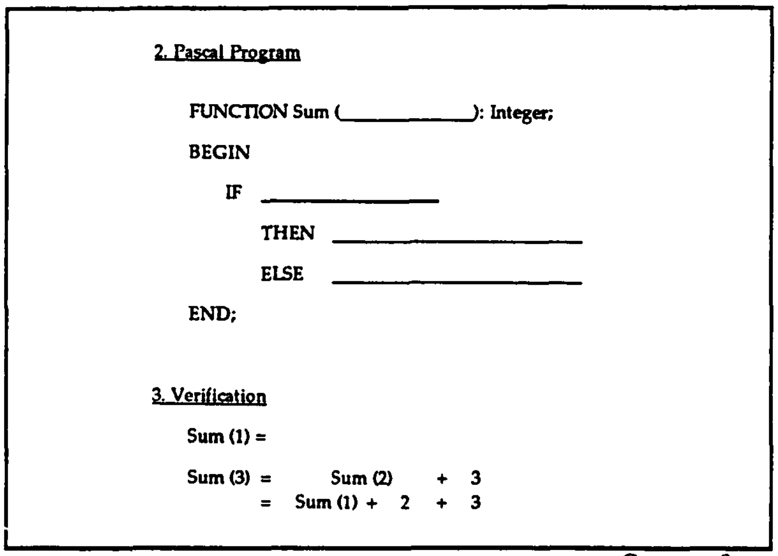
- Size ?
- Base Case ?
- Recursive Case ?

(Size steps toward Base Case?)

Sum (N) = 1, if N = 1

= Sum (N - 1) + N, if N > 1

Concrete 8



2. Pascal Program

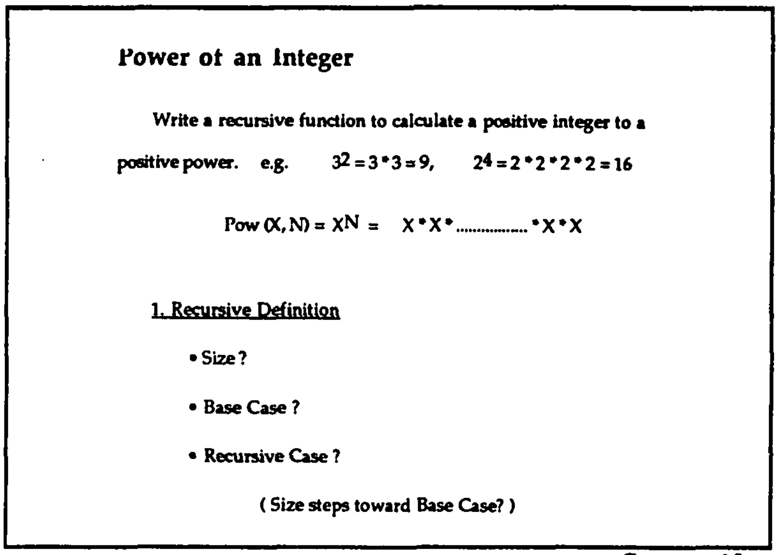
```
FUNCTION Sum (____________): Integer;
BEGIN
   IF ______________
      THEN ____________________
      ELSE ____________________
END;
```

3. Verification

```
Sum (1) =

Sum (3) =      Sum (2)    +  3
        =  Sum (1) +  2   +  3
```

Concrete 9



Power of an Integer

Write a recursive function to calculate a positive integer to a positive power. e.g. $3^2 = 3*3 = 9$, $2^4 = 2*2*2*2 = 16$

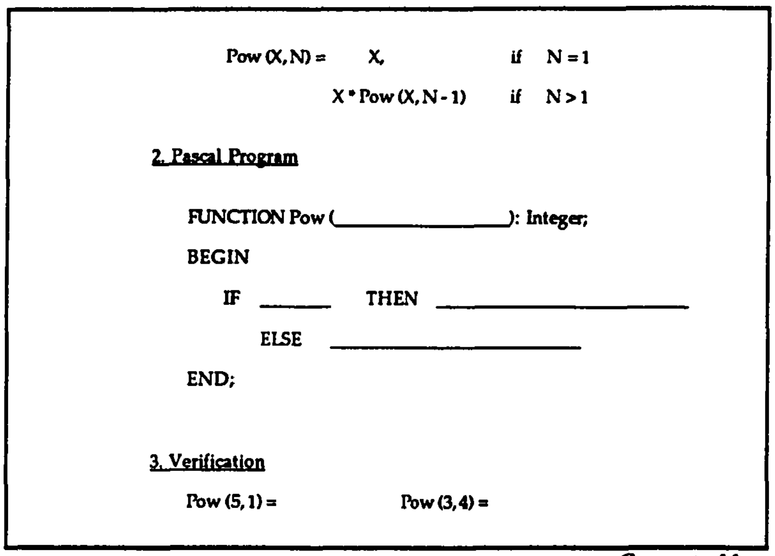
$$Pow(X, N) = X^N = X*X*\dots\dots\dots\dots*X*X$$

1. Recursive Definition

- Size?
- Base Case ?
- Recursive Case ?

(Size steps toward Base Case?)

Concrete 10



$$\text{Pow}(X,N) = \begin{cases} X, & \text{if } N = 1 \\ X * \text{Pow}(X, N-1) & \text{if } N > 1 \end{cases}$$

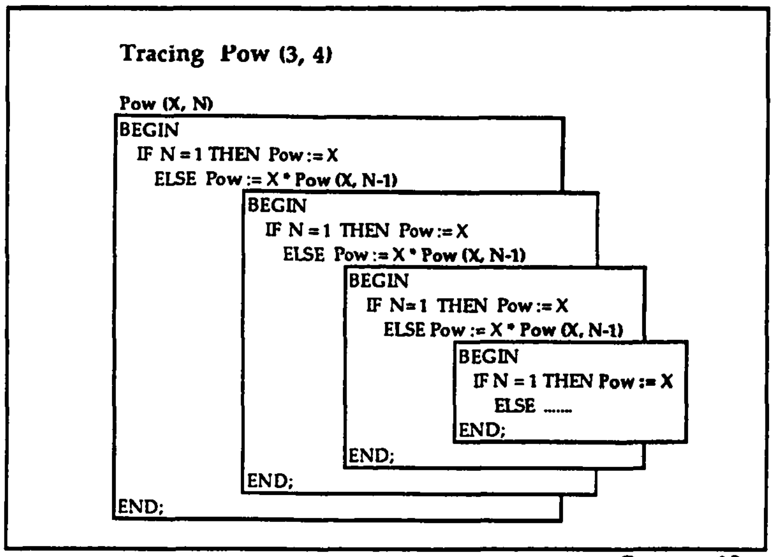
2. Pascal Program

```
FUNCTION Pow (_______________): Integer;
BEGIN
   IF ______   THEN ______________________
        ELSE ______________________
END;
```

3. Verification

Pow (5, 1) =   Pow (3, 4) =

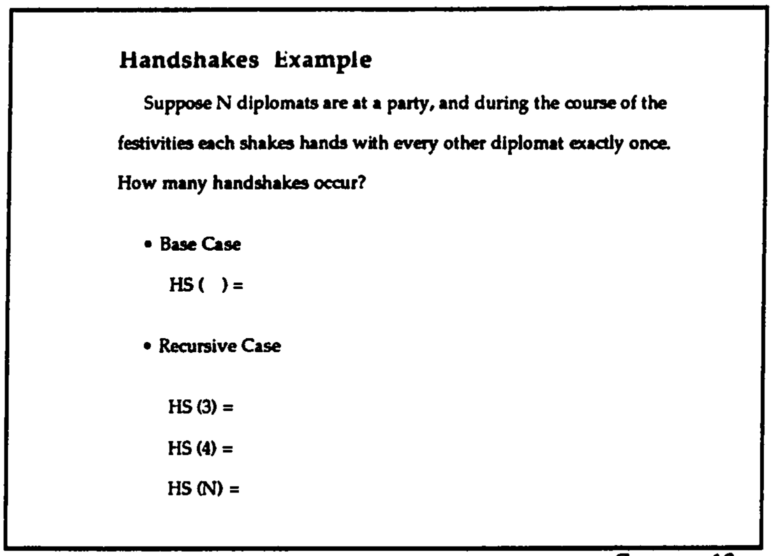
Concrete 11



Tracing Pow (3, 4)

```
Pow (X, N)
BEGIN
  IF N = 1 THEN Pow:=X
    ELSE Pow := X * Pow (X, N-1)
               BEGIN
                 IF N = 1 THEN Pow:=X
                   ELSE Pow := X * Pow (X, N-1)
                              BEGIN
                                IF N=1 THEN Pow:=X
                                  ELSE Pow := X * Pow (X, N-1)
                                              BEGIN
                                                IF N = 1 THEN Pow := X
                                                  ELSE ......
                                              END;
                              END;
               END;
END;
```

Concrete 12



Handshakes Example

Suppose N diplomats are at a party, and during the course of the festivities each shakes hands with every other diplomat exactly once. How many handshakes occur?

- Base Case

  HS ( ) =

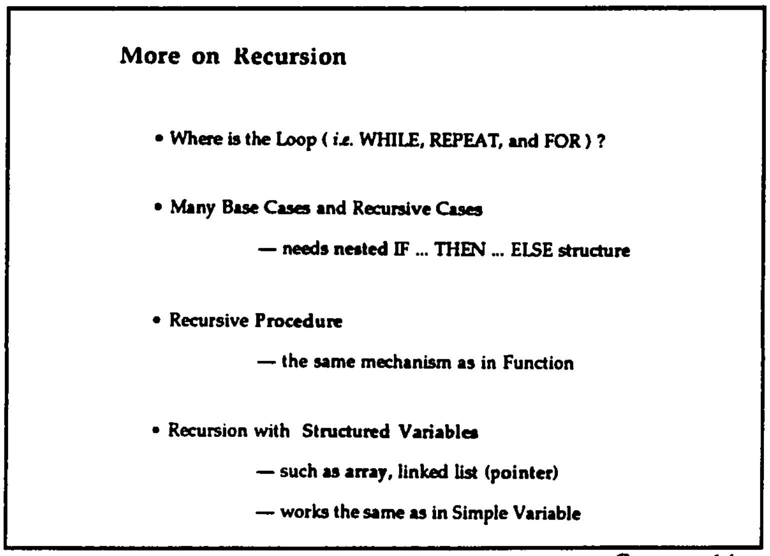
- Recursive Case

  HS (3) =

  HS (4) =

  HS (N) =

Concrete 13



More on Recursion

- Where is the Loop (*i.e.* WHILE, REPEAT, and FOR) ?
- Many Base Cases and Recursive Cases

  — needs nested IF ... THEN ... ELSE structure
- Recursive Procedure

  — the same mechanism as in Function
- Recursion with Structured Variables

  — such as array, linked list (pointer)

  — works the same as in Simple Variable

Concrete 14

Appendix H Abstract Instructional Material

Objectives

1. To understand the definition of recursion
2. To read and understand recursive programs (Recognition)
3. To generate base cases and recursive cases of a recursive function (Generalization)

1. Introduction

A. Use ***Mathematical Definition*** to introduce the concept of recursion.

B. Define the Base Case and Recursive Case

2. Recursion in Pascal

Example 1 **Factorial Problem**

Write a recursive function to calculate n!.

A. Show the solved recursive program

B. Identify the base case and recursive case

C. Use ***Mathematical Equation*** to trace the result of 3!

3. Designing Recursive Algorithms

A. Understand the problem

B. Determine the *size* of the problem to be solved

C. Identify the base case(s)

D. Identify the recursive case(s) using induction concept

4. Verification

Brief introduction of ***Mathematical Induction*** and use it to argue the correctness of the solved program:

1. The program is correct for the base case
2. It is also correct for the recursive case

5. Examples

Example 2 **Summing Problem**

A. Problem Specification: Write a recursive function to calculate the result of $1 + 2 + 3 + ... + N$.

B. Analysis and Design: Determine the size, base case(s), and recursive case(s) of the problem.

C. Implementation: Complete the recursive program.

D. Verification: First, trace the result of Sum(3) using ***Mathematical Equation***, then, argue the correctness of the algorithm by ***Mathematical Induction*** concept:

1. The algorithm is correct for the Base Case
2. The algorithm is correct for the Recursive Case

Example 3 **Power Problem**

A. Problem Specification: Write a recursive function to calculate a positive integer to a positive power. ($X^1 = X$; $X^N = X * X^{N-1}$)

B. Analysis and Design: Two parameters are required in this problem, which one is the size of the problem? Determine the base case(s) and recursive case(s).

C. Implementation: Complete the recursive program.

D. Verification: Arguing the correctness of the algorithm by ***Mathematical Induction*** concept:

1. Base case: Power(2,1)=2
2. Recursive Case: Power(2,N) = 2 * Power(2,N-1)

6. Elaboration and Conclusion

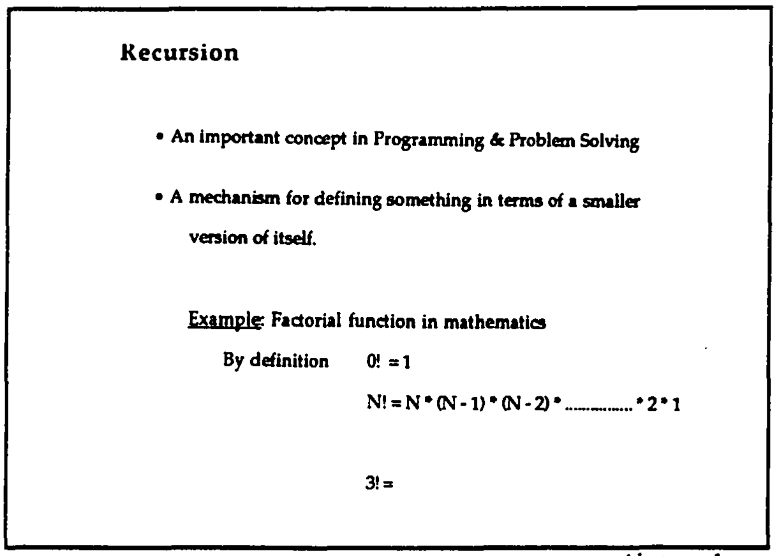
The mechanism of the recursion is the same in the following situations:

A. Many base cases and/or many recursive cases

B. Recursive Procedures

C. Structured Variables, e.g., arrays or linked list

Appendix I Transparencies for Abstract Instructional Material



Recursion

- An important concept in Programming & Problem Solving
- A mechanism for defining something in terms of a smaller version of itself.

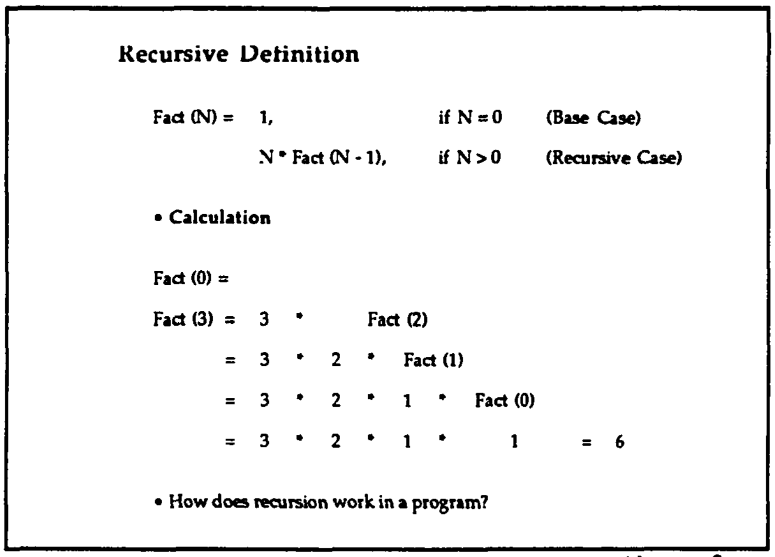
<u>Example</u>: Factorial function in mathematics

By definition $0! = 1$

$N! = N * (N - 1) * (N - 2) * \ldots\ldots\ldots\ldots * 2 * 1$

$3! =$

Abstract 1



Recursive Definition

Fact (N) =	1,	if N = 0	(Base Case)
	N * Fact (N - 1),	if N > 0	(Recursive Case)

- **Calculation**

Fact (0) =

Fact (3) = 3 * Fact (2)

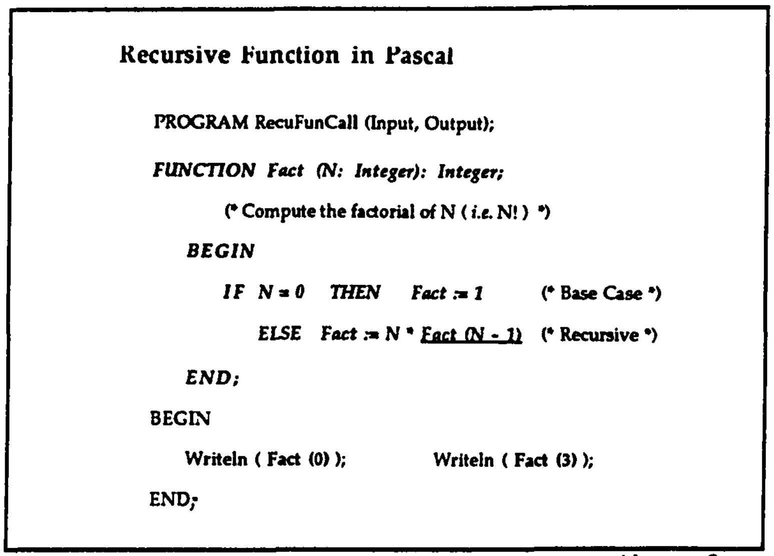
= 3 * 2 * Fact (1)

= 3 * 2 * 1 * Fact (0)

= 3 * 2 * 1 * 1 = 6

- How does recursion work in a program?

Abstract 2

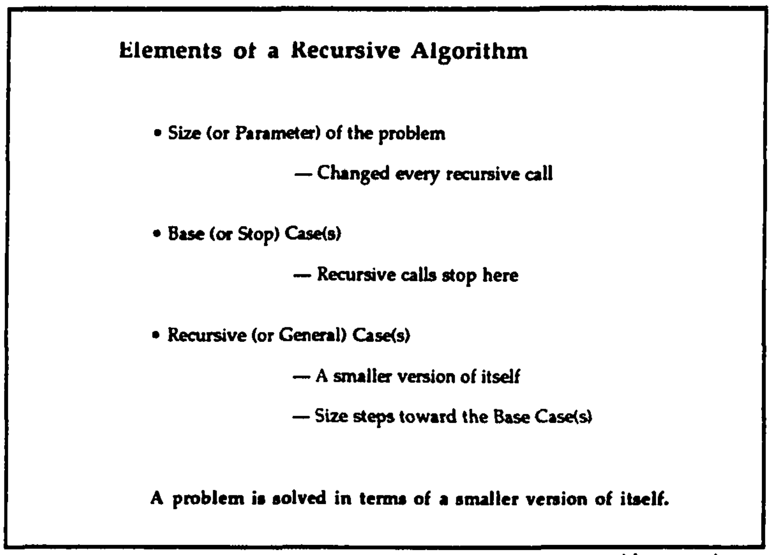


Recursive Function in Pascal

```
PROGRAM RecuFunCall (Input, Output);

FUNCTION Fact (N: Integer): Integer;
        (* Compute the factorial of N ( i.e. N! ) *)
    BEGIN
        IF  N = 0   THEN   Fact := 1          (* Base Case *)
            ELSE   Fact := N * Fact (N - 1)   (* Recursive *)
    END;
BEGIN
    Writeln ( Fact (0) );          Writeln ( Fact (3) );
END;
```

Abstract 3

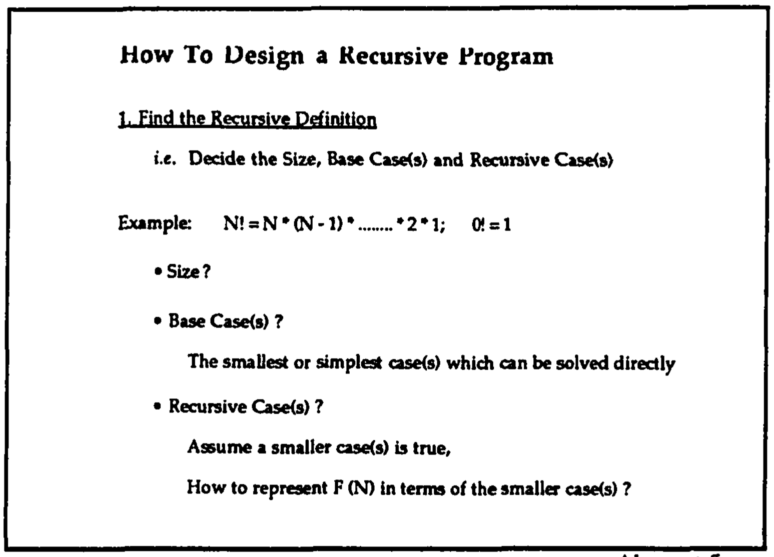


Elements of a Recursive Algorithm

- Size (or Parameter) of the problem
  - — Changed every recursive call
- Base (or Stop) Case(s)
  - — Recursive calls stop here
- Recursive (or General) Case(s)
  - — A smaller version of itself
  - — Size steps toward the Base Case(s)

A problem is solved in terms of a smaller version of itself.

Abstract 4



How To Design a Recursive Program

1. Find the Recursive Definition

i.e. Decide the Size, Base Case(s) and Recursive Case(s)

Example: N! = N * (N - 1) * .... * 2 * 1; 0! = 1

- Size?
- Base Case(s)?

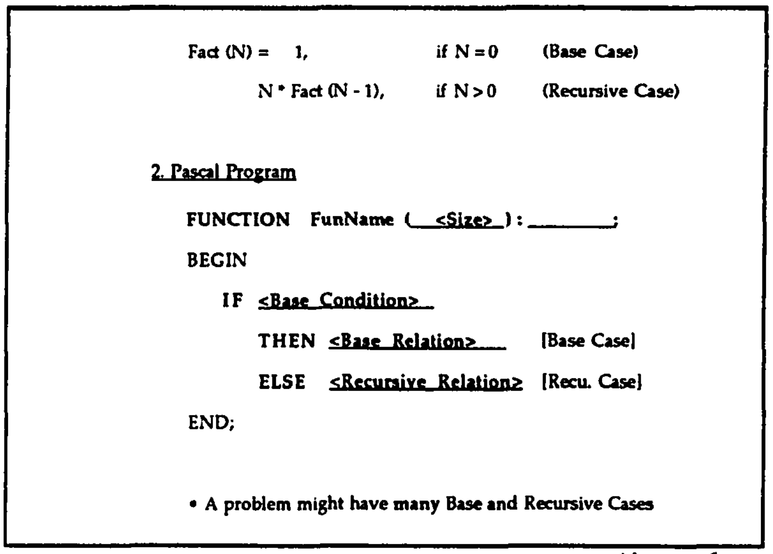
  The smallest or simplest case(s) which can be solved directly

- Recursive Case(s)?

  Assume a smaller case(s) is true,

  How to represent F (N) in terms of the smaller case(s)?

Abstract 5



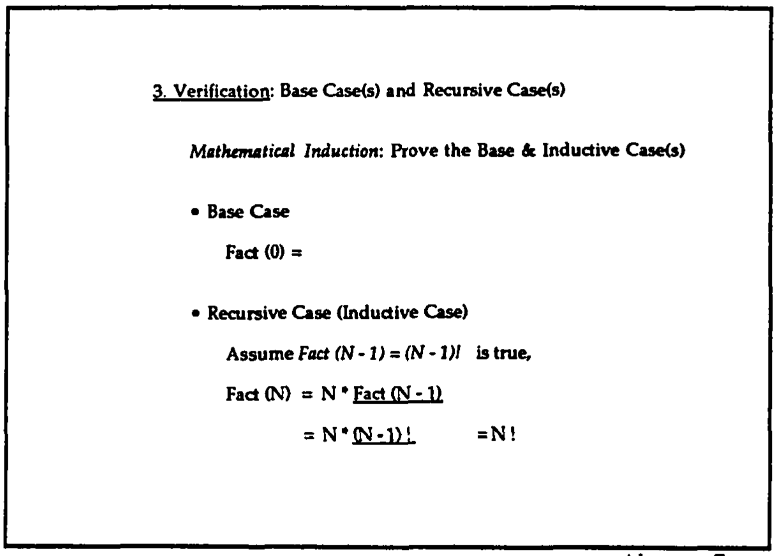
```
Fact (N) =  1,                if N = 0    (Base Case)
            N * Fact (N - 1), if N > 0    (Recursive Case)
```

2. Pascal Program

```
FUNCTION  FunName ( <Size> ) : ________;
BEGIN
   IF  <Base Condition>
       THEN  <Base Relation>        [Base Case]
       ELSE  <Recursive Relation>   [Recu. Case]
END;
```

- A problem might have many Base and Recursive Cases

Abstract 6



3. Verification: Base Case(s) and Recursive Case(s)

Mathematical Induction: Prove the Base & Inductive Case(s)

- Base Case

  Fact (0) =

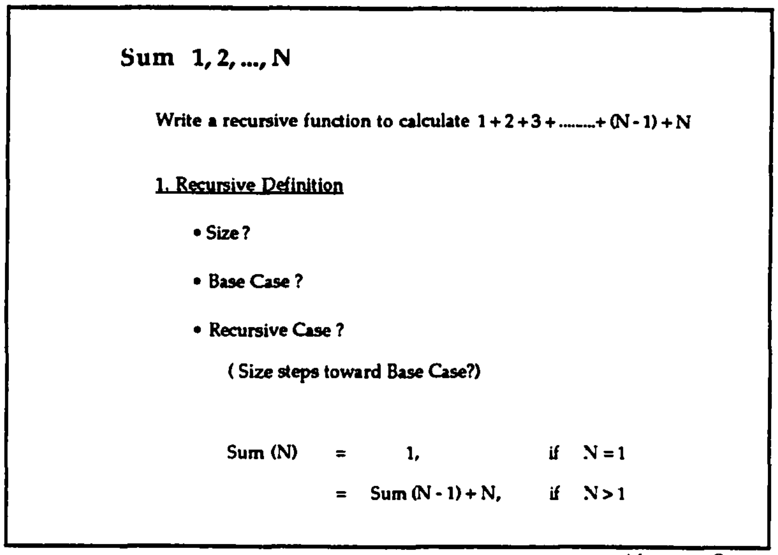
- Recursive Case (Inductive Case)

  Assume *Fact (N - 1) = (N - 1)!* is true,

  Fact (N) = N * Fact (N - 1)

  = N * (N - 1) ! = N !

Abstract 7



Sum 1, 2, ..., N

Write a recursive function to calculate 1 + 2 + 3 + ...+ (N - 1) + N

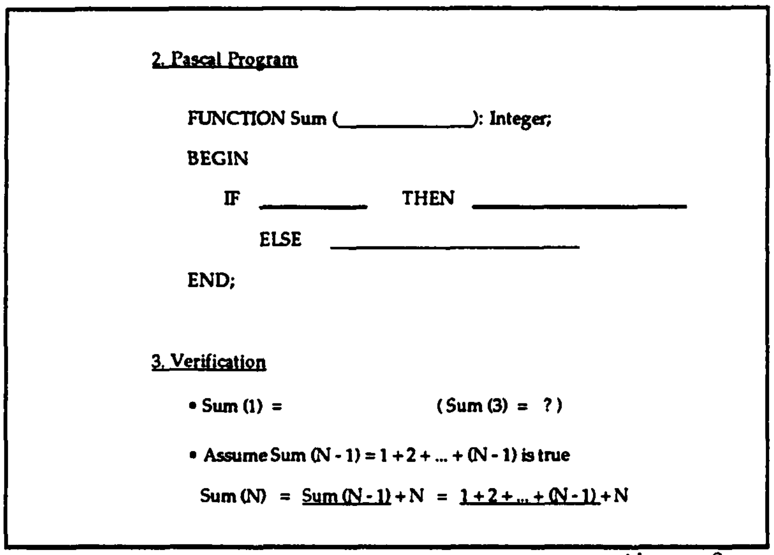
1. Recursive Definition

- Size ?
- Base Case ?
- Recursive Case ?

  (Size steps toward Base Case?)

$$\text{Sum}(N) = \begin{cases} 1, & \text{if } N = 1 \\ \text{Sum}(N-1) + N, & \text{if } N > 1 \end{cases}$$

Abstract 8



2. Pascal Program

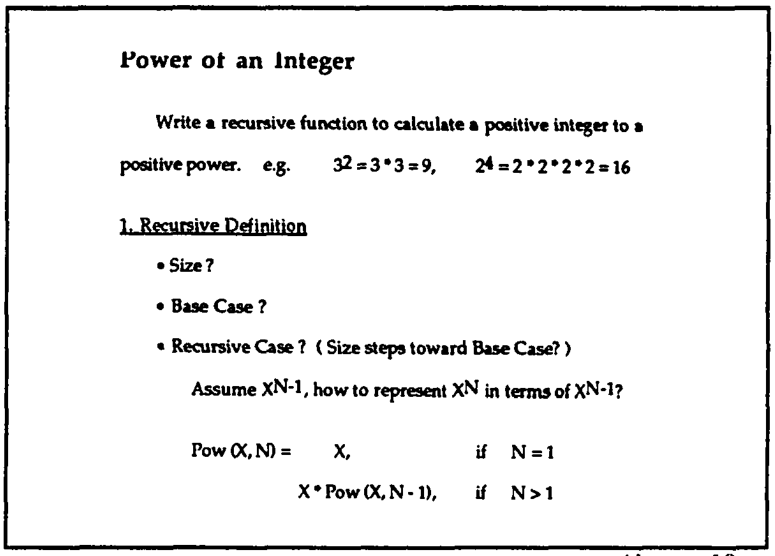
```
FUNCTION Sum (____________): Integer;
BEGIN
   IF _________ THEN __________________
       ELSE ____________________
END;
```

3. Verification

- Sum (1) = (Sum (3) = ?)
- Assume Sum (N - 1) = 1 + 2 + ... + (N - 1) is true

$\text{Sum (N)} = \underline{\text{Sum (N - 1)}} + N = \underline{1 + 2 + \ldots + (N - 1)} + N$

Abstract 9



Power of an Integer

Write a recursive function to calculate a positive integer to a positive power. e.g. $3^2 = 3 * 3 = 9$, $2^4 = 2 * 2 * 2 * 2 = 16$

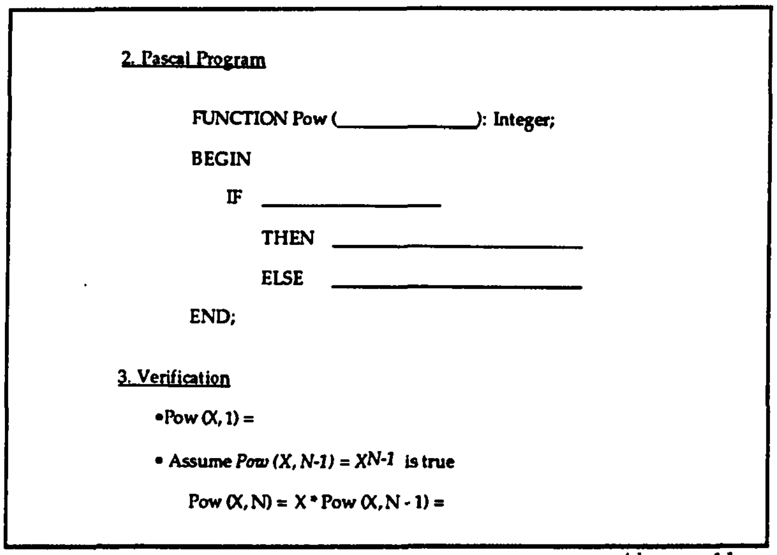
1. Recursive Definition

- Size ?
- Base Case ?
- Recursive Case ? (Size steps toward Base Case?)

Assume X^{N-1}, how to represent X^N in terms of X^{N-1}?

$$\text{Pow}(X, N) = \begin{cases} X, & \text{if } N = 1 \\ X * \text{Pow}(X, N - 1), & \text{if } N > 1 \end{cases}$$

Abstract 10



2. Pascal Program

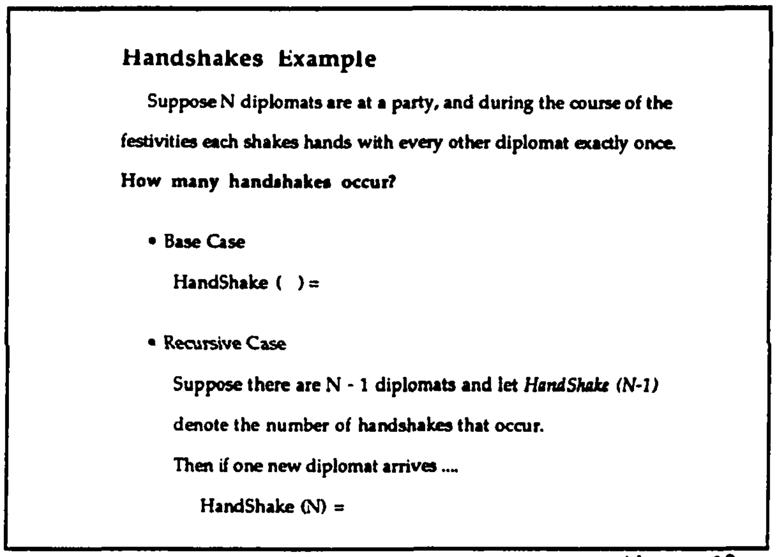
```
FUNCTION Pow (____________): Integer;
BEGIN
   IF ____________
      THEN ________________
      ELSE ________________
END;
```

3. Verification

- Pow (X, 1) =
- Assume *Pow (X, N-1)* = X^{N-1} is true

  Pow (X, N) = X * Pow (X, N - 1) =

Abstract 11



Handshakes Example

Suppose N diplomats are at a party, and during the course of the festivities each shakes hands with every other diplomat exactly once. **How many handshakes occur?**

- Base Case

  HandShake (   ) =

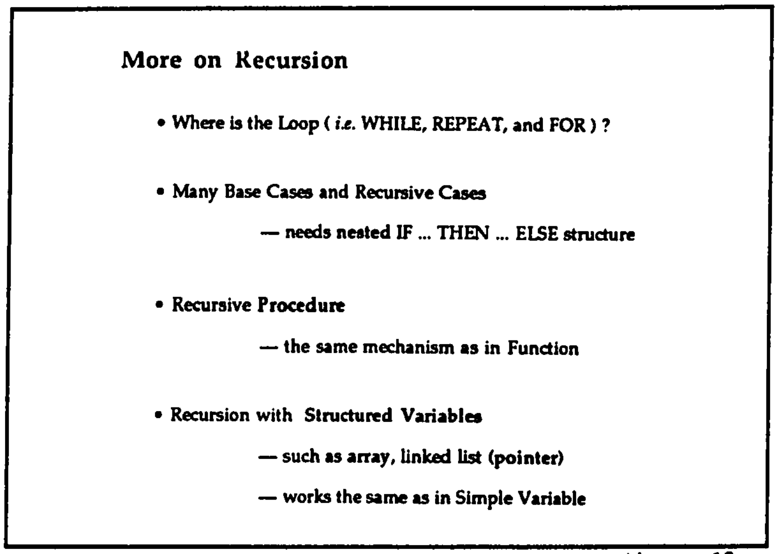
- Recursive Case

  Suppose there are N - 1 diplomats and let *HandShake (N-1)* denote the number of handshakes that occur.

  Then if one new diplomat arrives ....

  HandShake (N) =

Abstract 12



More on Recursion

- Where is the Loop (*i.e.* WHILE, REPEAT, and FOR) ?
- Many Base Cases and Recursive Cases
  - — needs nested IF ... THEN ... ELSE structure
- Recursive Procedure
  - — the same mechanism as in Function
- Recursion with Structured Variables
  - — such as array, linked list (pointer)
  - — works the same as in Simple Variable

Abstract 13

Appendix J Pretest

CS 304P - Review Test I - Fall 92

October 1, 1992

NOTE: Correct responses are based on standard Pascal as presented in the textbook. Record your answers on scantron form.

Multiple choice - choose the single best answer for each item below. (2 points each)

1. Every Pascal program must include:
   a. BEGIN and END
   b. input and output
   c. at least one constant
   d. at least one variable
   e. all of the above

2. The Pascal data type which can represent only positive numbers is
   a. char
   b. integer
   c. real
   d. boolean
   e. none of the above

3. Which of the following is a legal Pascal identifier for a variable?
   a. VALUE&
   b. big-number
   c. program
   d. Seco2d
   e. none of the above

4. Given two integers as input, which of the following operators does not return an integer result?
   a. +
   b. -
   c. *
   d. /
   e. none of the above

5. The part of a computer that stores both programs and data is the
   a. CPU
   b. control unit
   c. memory
   d. software
   e. interface

Given the boolean variables A, B, C, and D, show the values of expressions 6-9, if: A = FALSE, B = FALSE, C = TRUE, D = TRUE.

6. **(A OR B) AND (C OR D)**
   a. TRUE
   b. FALSE
   c. compile-time error
   d. run-time error
   e. not enough information

7. **NOT A AND B AND (32 > 51)**
   a. TRUE
   b. FALSE
   c. compile-time error
   d. run-time error
   e. not enough information

8 . **(7 > 4 OR 3) AND A**
   a. TRUE
   b. FALSE
   c. compile-time error
   d. run-time error
   e. not enough information

9. Which of the following CANNOT be a computer output device?
   a. keyboard
   b. magnetic tape drive
   c. liquid crystal display terminal
   d. laser printer
   e. video display terminal

10. A step-by-step finite process for solving a problem is an
   a. implementation
   b. induction
   c. altercation
   d. algorithm
   e. extension

11. Given the declaration **VAR E: BOOLEAN;** What is the value of the expression **(E OR NOT E)** ? (Assume E has been initialized).
   a. TRUE
   b. FALSE
   c. E
   d. NOT E
   e. not enough information

12. What is written by the statement ('_' denotes a space):
    WRITE (SQR (0.32/2.0):5:4)
    a. 0.0256
    b. _0.026
    c. _0.0256
    d. _0.03
    e. error

13. Which of the following is a correct Pascal expression equivalent to:

$$\frac{(5 - 3Y)}{4} \times \frac{1}{Y - 5}$$

    a. (5 - Y + Y + Y DIV 4) DIV (Y - 5)
    b. (5 - 3) * Y / 4 * (1/ (Y - 5))
    c. ((5 - (3 (Y))) / 4) * 1 / (Y - 5)
    d. ((5 - (Y + (Y + (Y)))) / 4) / (Y - 5)
    e. none of the above

14. Given the following declarations, which of the choices is a correct Pascal statement?

```
VAR   I1, I2:      INTEGER;
      R1, R2:      REAL;
      C1, C2:      CHAR;
      B1, B2:      BOOLEAN;
```

    a. B1 AND B2 < 3 = 4
    b. C1 AND 'A' OR R1 < R2
    c. R1 < 3 + I1 = B2 OR NOT B1
    d. (C1 => 'M') AND B2
    e. all of the above

15. Which operator is evaluated **FIRST** in the following Pascal expression?
    A AND (B OR (X = Y)) AND (Z < 49)
    a. the first AND
    b. OR
    c. =
    d. the second AND
    e. <

Questions 16-19 are based on the following program. This code counts the question marks found in the input file, DATA.

```
PROGRAM TESTI (INPUT, OUTPUT, DATA);
VAR DATA: TEXT; CH: CHAR; NUM: INTEGER;
BEGIN
  ____1____
  NUM := 0;
  WHILE ____2____ DO
     BEGIN
       WHILE ____3____ DO
          BEGIN
              READ (DATA, CH);
              IF CH = '?' THEN NUM := NUM + 1
          END;
       ____4____
     END
END.
```

16. Which of the following properly fills blank 1 above?
    a. REWRITE(DATA);
    b. RESET(DATA);
    c. CH := '?';
    d. RESET(INPUT);
    e. none of the above

17. Which of the following properly fills blank 2 above?
    a. NOT EOF
    b. NOT EOLN
    c. CH = '?'
    d. NOT EOF(DATA)
    e. none of the above

18. Which of the following properly fills blank 3 above?
    a. NOT EOF(DATA)
    b. NOT EOLN(DATA)
    c. NOT (CH = '?')
    d. NUM = 0
    e. none of the above

19. Which of the following properly fills blank 4 above?
    a. READ(DATA);
    b. READLN(DATA);
    c. RESET(DATA);
    d. REWRITE(DATA);
    e. none of the above

20. Which operator is evaluated **LAST** in the following Pascal expression?

    (12.5 * 3 - 42) / (19 + 432 DIV 66)

    a. *
    b. -
    c. /
    d. +
    e. DIV

21. Which of the following is a correct Pascal equivalent of **A < B**?
    a. NOT (A >= B)
    b. A <> B
    c. NOT (A > B)
    d. A NOT > B
    e. all of the above

22. Which of the following is a correct Pascal condition requiring both X and Y to be at least 27?
    a. X AND Y > 27
    b. X > 27 AND Y > 27
    c. NOT (X < 27) OR NOT (Y < 27)
    d. NOT ((X < 27) OR (Y < 27))
    e. none of the above

23. What are the values of the variables after execution of the following statements? L, M, N and P are integer variables.

```
L := 12 + 5;
M := L - 8;
N := L + M * 2;
M := 17;
P := L + M + N;
```

    a. L is 17; M is 9; N is 68; P is 94
    b. L is 17; M is 26; N is 69; P is 112
    c. L is 17; M is 17; N is 34; P is 68
    d. L is 17; M is 17; N is 35; P is 69
    e. error

24. The purpose of testing should be to show
    a. that the program runs correctly on the class data
    b. the absence of errors
    c. the presence of errors
    d. that the program compiles correctly
    e. none of the above

Given the declarations, read statements, and file, DATA, below, what is the value of the variables after executing the statements in numbers 25-28?

```
DECLARATIONS:                          DATA (each _ represents a blank space):
      VAR num1, num2 : INTEGER;              23_9B_1
          ch1, ch2 : CHAR;                   FR_725
          DATA : TEXT;                             _96_N_34
```

25.
```
RESET(DATA);
READ (DATA, num1, num2, ch1, ch2);
```
a. num1 = 2, num2 = 3, ch1 = '_', ch2 = '9'
b. num1 = 23, num2 = 9, ch1 = 'B', ch2 = '_'
c. num1 = 23, num2 = 9B, ch1 = '_', ch2 = '1'
d. num1 = 23, num2 = 9, ch1 = 'F', ch2 = 'R'
e. error

26.
```
RESET(DATA);
READ (DATA, ch1);
READLN (DATA);
READ (DATA, ch2, num1, num2 );
```
a. num1 = 9, num2 = 1, ch1 = '2', ch2 = 'F'
b. num1 = 9, num2 = 1, ch1 = '2', ch2 = '3'
c. num1 = 725, num2 = 96, ch1 = '23', ch2 = '_'
d. num1 = 23, num2 = 9, ch1 = 'B', ch2 = '_'
e. error

27.
```
RESET(DATA);
READLN (DATA, ch1, ch2, num1);
READ (DATA, ch2, ch1);
READ (DATA, num1, num2);
```
a. num1 = 1, num2 = 725, ch1 = '2', ch2 = '3'
b. num1 = 725, num2 = 96, ch1 = 'R', ch2 = 'F'
c. num1 = 96, num2 = 725, ch1 = 'F', ch2 = 'R'
d. num1 = 96, num2 = 34, ch1 = '_', ch2 = '_'
e. error

28.
```
RESET (DATA);
READLN (DATA, num1, ch1);
READLN (DATA, ch1, ch2, num2);
READ (DATA, ch1, num1, ch2, ch1, ch2);
```
a. num1 = 96, num2 = 725, ch1 = 'N', ch2 = '_'
b. num1 = 23, num2 = 725, ch1 = '_', ch2 = 'R'
c. num1 = 23, num2 = 725, ch1 = 'F', ch2 = '4'
d. num1 = 96, num2 = 725, ch1 = '3', ch2 = '4'
e. error

What is printed by the following code, given the values in 29-35.

```
IF X <= Y THEN
        IF X < Y THEN
               WRITELN ('RED')
        ELSE
        WRITELN('BLUE')
ELSE IF X < Y THEN
        WRITELN('GREEN')
        ELSE
        WRITELN('YELLOW')
```

29. X = 12, Y = 34
    a. BLUERED
    b. RED
    c. GREEN
    d. YELLOW
    e. none of the above

30. X = 6, Y = -9
    a. RED
    b. GREEN
    c. YELLOW
    d. GREENYELLOW
    e. none of the above

31. X = 51, Y = 51
    a. RED
    b. BLUE
    c. GREEN
    d. BLUEYELLOW
    e. none of the above

32. Given the following program, which line contains code which will cause a compiler error?

```
1     PROGRAM REVIEWI (INPUT, OUTPUT);
2     VAR A, B : REAL;
3     BEGIN
4             A := 7;  B := 6.25;
5             IF A + B > 10
6                    THEN WRITELN ('BIG NUMBERS');
7                    ELSE WRITELN ('LITTLE NUMBERS')
8     END.
```

a. line 2
b. line 4
c. line 6
d. line 8
e. all of the above

33. What is the proper loop invariant for the following code segment?

```
Count := 1;
Sum := 0;
WHILE Count <= B DO
        BEGIN
                Sum := Sum + A;
                Count := Count + 1
        END;
```

a. Sum = (Count - 1) * A
b. Count >= 1 AND Count <= B + 1
c. (Sum = (Count -1) * A) AND (Count >= 1 AND Count <= B + 1)
d. (Sum = (Count -1) * A) AND (Count >= 1 AND Count <= B)
e. none of the above

34. Which of the following statements sets CubeEven to TRUE if the cube of Number is even and FALSE otherwise?

a.
```
IF ((Number * Number * Number MOD 2) = 0) = TRUE
        THEN CubeEven := TRUE
        ELSE CubeEven := FALSE;
```
b.
```
IF ((Number * Number * Number) MOD 2 = 0)
        THEN CubeEven := TRUE
        ELSE CubeEven := FALSE;
```
c. CubeEven := (Number * Number * Number MOD 2 = 0);
d. all of the above
e. none of the above

35. Which of the following is syntactically invalid?
(All variables are integers and have been initialized)

1. A := Constant;
2. B := 5.5 MOD 6;
3. Cat := Cat + Dog;

a. 1 only
b. 2 only
c. 3 only
d. 1 and 2 only
e. all are valid

Appendix K Posttest

Name ____________________ ID# ______________________

Have you studied recursion before? ______Yes ______No

Please show your work

Question 1

Consider the Pascal function described below:

```
FUNCTION F (N: Integer) : Integer;
BEGIN
    IF  N = 0
        THEN F := 1
        ELSE      F := F (N - 1) + 2
END;
```

1. What is the value of F (1)? ________
2. What is the value of F (3)? ________

Question 2

Complete the following *recursive* function which performs multiplication using addition. (e. g., 5 X 3 = 5 + 5 + 5)

```
FUNCTION Multiply (P, Q: Integer): Integer;
    (*  PreCondition:     P and Q are defined and Q > 0      *)
    (*  PostCondition:    Returns P X Q                      *)
BEGIN
    IF ____________________
        THEN ______________________________
        ELSE     _________________________________
END;
```

Question 3

What is the output of the following Pascal procedure?

```
PROCEDURE PrintNum (N: Integer);
BEGIN
   IF  N = 0
       THEN (* do nothing *)
       ELSE  BEGIN
                PrintNum (N - 1);
                Write (N)
             END
END;
```

1. What is the output of PrintNum (1)? ____________________

2. What is the output of PrintNum (3)? ____________________

Question 4

Complete the following *recursive* function which generates the Nth number in the Fibonacci sequence, which is defined to be : 1, 1, 2, 3, 5, 8, 13, ...

```
FUNCTION Fib (N: Integer): Integer;
   (*  PreCondition:     N is defined and N >= 1                      *)
   (*  PostCondition:    Returns the Nth number of the sequence       *)
BEGIN
   IF  ________________

       THEN Fib := 1

       ELSE    ____________________________
END;
```

Appendix L Retention Test 1

1. Given the following function, what is the returned value of F(4)?

```
FUNCTION F (N: Integer) : Integer;
BEGIN
   IF  N = 0
       THEN     F := 1
       ELSE     F := F (N - 2) + 1
END;
```

   a. 1
   b. 2
   c. 3
   d. 4
   e. none of the above

2. The following function calculates the sum of *successive even integers* starting at 0 and ending at N (N is an even integer).
(for example SUM(6) = 12, (6 + 4 + 2 + 0))

```
FUNCTION Sum (N: Integer): Integer;
BEGIN (* Sum *)
   IF  N = 0
       THEN     Sum := 0
       ELSE     Sum := ______________________
END;
```

Which of the following properly fills the blank above?

   a. Sum (N) + 2
   b. Sum (N - 1) + 2
   c. Sum (N - 1) + N
   d. Sum (N - 2) + 1
   e. Sum (N - 2) + N

3. Given the following function, what value is returned by Fib (5)?

```
FUNCTION Fib (N: Integer): Integer;
BEGIN
   IF  (N = 1) OR (N = 2)
      THEN     Fib := 1
      ELSE     Fib := Fib (N - 1) + Fib (N - 2)
   END;
```

a. 5
b. 3
c. 2
d. 1
e. none of the above

Questions 4 and 5 refer to the type definition and *recursive* function below which calculates the value of a positive integer to a *non-positive* power. Note $X^0 = 1$; and $X^N = 1/X^{-N}$, if $N < 0$. (for example $X^{-5} = 1/X^5$; $2^{-3} = 1/2^3 = 8$)

```
TYPE Negatives = -MaxInt..0;
FUNCTION NegPower (X: Integer; N: Negatives): Real;
BEGIN
   IF  N = 0
      THEN     NegPower :=______1__________
      ELSE     NegPower :=______2__________
   END;
```

4. Which of the following properly fills blank 1 above?

a. 0
b. 1
c. -1
d. X
e. NegPower (X, N - 1)

5. Which of the following properly fills blank 2 above?
    a. NegPower (X, N - 1)
    b. NegPower (X, N - 1) * X
    c. NegPower (X, N + 1) * X
    d. (NegPower (X, N - 1) / X)
    e. (NegPower (X, N + 1) / X)

Appendix M Retention Test 2

1. Given the following function, what is the value returned by F(-2)?

```
FUNCTION   F (N: Integer): Integer;
    BEGIN
        IF  N = 0
            THEN      F := 1
            ELSE      F := F (N + 1) + 1
    END;
```

a. -3
b. 1
c. 2
d. 3
e. none of the above

2. Given the following function, what is the value returned by F(3, 2)?

```
FUNCTION   F (P, Q: Integer): Integer;
    BEGIN
        IF  Q = 1
            THEN      F := P
            ELSE      F := P + F(P, Q-1)
    END;
```

a. 3
b. 5
c. 6
d. 9
e. none of above

Questions 3 and 4 refer to the following function which generates the Nth integer the sequence: 0, 3, 6, 9, 12, ...(For example, GenNum(1) generates 0; and GenNum(3) generates 6)

```
FUNCTION GenNum (N : Integer) : Integer;
    BEGIN
        IF  N = 1
            THEN      GenNum := _______1_________
            ELSE      GenNum := _______2_________
    END:
```

3. Which of the following properly fills the blank 1 above?
   a. 0
   b. 1
   c. 3
   d. GenNum(N-1)
   e. GenNum(N-3)

4. Which of the following properly fills the blank 2 above?
   a. GenNum(N-1) + GenNum(N-2)
   b. GenNum(N-1) + N
   c. GenNum(N-3) + N
   d. GenNum(N-1) + 3
   e GenNum(N-3) + 3

5. The following procedure RevPrintArray writes out all the elements in an array in reverse order.

```
TYPE        ArrayType = Array [1..Length] of Integer;

PROCEDURE RevPrintArray (A: ArrayType; Length: Integer);
   BEGIN
      IF Length > 0  THEN
         BEGIN
            ______________________________
         END
   END;
```

Which of the following properly fills the blank above?

a. Writeln(A[1]); RevPrintArray(A, Length-1)
b. Writeln(A[Length]); RevPrintArray(A, Length-1)
c. RevPrintArray(A, Length-1); Writeln(A[Length])
d. RevPrintArray(A, Length); Writeln(A[Length-1])
e. none of the above

Bibliography

Aho, A. V., & Ullman, J. D. (1992). *Foundations of computer science*. New York, NY: W. H. Freeman and Company.

Allinson, C. W., & Hayes, J. (1990). Validity of the Learning Style Questionnaire. *Psychological Reports, 67*, 859-866.

Anderson, J. R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8*, 87-129.

Anzai, Y., & Uesato, Y. (1982). Learning recursive procedures by middle school children. *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, 100-102.

Atkinson, G. Jr., Murrell, P. H., & Winters, M. R. (1990). Career personality types and learning styles. *Psychological Reports, 66*, 160-162.

Atkinson, G., Jr. (1988). Reliability of the Learning Style Inventory 1985. *Psychological Reports, 62*, 755-758.

Atkinson, G., Jr. (1989). Kolb's Learning Style Inventory 1985: test-retest deja vu. *Psychological Reports, 64*, 991-995.

Ausubel, D. P. (1968). *Educational Psychology: a Cognitive View*. New York: Holt, Rinehart & Winston.

Ausubel, D. P., Novak, J. D., & Hanesian, H. (1978). *Educational Psychology: a Cognitive View* (2nd ed.). New York: Holt, Rinehart & Winston.

Bennett, K. B. (1984). The effect of display design on the user's mental model of a perceptual database system. (Doctoral dissertation, The Catholic University of America, 1984). *Dissertation Abstracts International, 45*, 1604B.

Bhuiyan, S. H., Greer, J. E., & McCalla, G. I. (1989). Mental models of recursion and their use in the SCENT programming advisor. In S. Ramani, R. Chandrasekar, & K. R. S. Anjaneyulu (Eds.), *Knowledge-Based Computer Systems, KBCS 89* (pp. 135-144). Bombay, India.

Bhuiyan, S. H., Greer, J. E., & McCalla, G. I. (1991). *Characterizing, Rationalizing, and Reifying Mental Models of Recursion.* SK., Canada: University of Saskatchewan, Laboratory for Advanced Research in Intelligent Educational Systems.

Bonar, J., & Soloway, E. (1985). Pre-programming knowledge: a major source of misconceptions in novice programmers. *Human-Computer Interaction, 1*, 133-161.

Borgman, C. L. (1984). The User's Mental Model of an Information Retrieval System: Effects on Performance. (Doctoral dissertation, Stanford University, 1983). *Dissertation Abstracts International, 45*, 4-AB.

Bostrom, R. P., Olfman, L., & Sein, M. K. (1987). The importance of individual differences in end-user training: The case for learning style. *Proceedings of the 1988 ACM SIGCPR Conference on the Management of Information Systems Personnel*, 133-141.

Bowman, B. C., & Seagraves, K. (1985). Picturing recursion. *The Computing Teacher, 12*(7), 28-32.

Brewer, W. F. (1987). Schemas versus mental models in human Memory. In P. Morris (Ed.), *Modeling Cognition* (pp. 187-197). New York: John Wiely & Sons.

Carrier, C. A., Williams, M. D., & Dalgaard, B. F. (1988). College students' perceptions of notetaking and their relationship to selected learner characteristics and course achievement. *Research in Higher Education, 28*(3), 223-239.

Carroll, J. M., & Olson, J. R. (Eds.). (1987). *Mental Models in Human-Computer Interaction: Research Issues about What the User of Software Knows.* Washington, DC: National Academy Press.

Catalanello, R. & Bremenstuhl, D. (1978). An investigation of innovative teaching methodologies. *Academy of Management Proceedings*, 18-22.

Cavaiani, T. P. (1989). Cognitive style and diagnostic skills of student programmers. *Journal of Research on Computing in Education, 22*, 411-420.

Collins, A. (1985). Component models of physical systems. *Proceedings of the Seventh Annual Conference of the Cognitive Society*, 80-89.

Corman, L. S. (1986). Cognitive style, personality type, and learning ability as factors in predicting the success of the beginning programming student. *SIGCSE Bulletin, 18*(4), 80-89.

Cornwell, J. M., Manfredo, P. A., & Dunlap, W. P. (1991). Factor analysis of the 1985 revision of Kolb's Learning Style Inventory. *Educational and Psychological Measurement, 51*(2), 455-463.

Dale, N. B., & Lilly, S. C. (1991). *Pascal Plus Data Structures* (3rd ed.). Lexington, MA: D. C. Heath.

Dale, N. B., & Weems, C. (1991). *Pascal* (3rd ed.). Lexington, MA: D. C. Heath.

Davidson, G. V. (1990). Matching learning styles with teaching styles: Is it a useful concept? *Performance and Instruction,* 29(4), 36-38.

Davidson, G. V., Savenye, W. C., & Orr, K. B. (1992). How do learning styles relate to performance in a computer applications course? *Journal of Research on Computing in Education, 24*(3), 348-358.

De Kleer, J., & Brown, J. S. (1981). Mental models of physical mechanisms and their acquisition. In J. R. Anderson (Ed.), *Cognitive Skills and their Acquisition* (pp. 285-309). Hillsdale, NJ: Erlbaum.

Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research,* 2(1), 57-73.

Du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box: presenting computing concepts to novices. *International Journal of Man-Machine Studies, 14,* 237-249.

Duit, R. (1991). On the role of analogies and metaphors in learning science. *Science Education, 75*(6), 649-672.

Eliot, J., Lovell, K., Dayton, C. M., & McGrady, B. F. (1979). A further investigation of children's understanding of recursive thinking. *Journal of Experimental Child Psychology, 28,* 149-157.

Er, M. C. (1984). On the complexity of recursion in problem-solving. *International Journal of Man-Machine Studies, 20,* 537-544.

Ferrel, B. G. (1983). A factor analytic comparison of four learning styles instruments. *Journal of Educational Psychology, 75*(1), 33-39.

Ford, G. (1982). A framework for teaching recursion. *SIGCSE Bulletin, 14*(2), 32-39.

Ford, G. (1984). An implementation-independent approach to teach recursion. *SIGCSE Bulletin, 16*(1), 213-216.

Foss, D. J., Rosson, M. B., & Smith, P. L. (1982). Reducing manual labor: An experimental analysis of learning aids for a text editor. *Proceedings of the CHI '82 Conference on Human Factors in Computer Systems*, 332-336.

Galletta, D. F. (1986). A Learning Model of Information Systems: The effects of Orienting Materials, Ability, Expectations and Experience on Performance, Usage and Attitudes. (Doctoral Dissertation, University of Minnesota, 1985). *Dissertation Abstracts International, 46*, 2008A.

Geiger, M. A. (1991). Performance during the first year of college: Differences associated with learning styles. *Psychological Reports, 68*, 633-634.

Gentner, D. (1983). Structure-mapping: a theoretical framework for analogy. *Cognitive Science, 7*, 155-170.

Gentner, D. (1988). Analogical inference and analogical access. In A. Prieditis (Ed.), *Analogica* (pp. 63-88). Las Altos, CA: Morgan Kaufmann.

Gentner, D., & Gentner, D. R. (1983). Flowing waters or teeming Crowds: mental models of electricity. In D. Gentner & A. L. Stevens (Eds.), *Mental Models* (pp. 99-129). Hillsdale, NJ: Erlbaum.

Greer, J. E. (1987). Empirical comparison of techniques for teaching recursion in introductory computer science. (Doctoral dissertation, The University of Texas at Austin, 1987). *Dissertation Abstracts International, 48*, 1415B.

Gregorc, A. F. (1984). *Gregorc Style Delineator: Development Technical and Administration Manual*. Columbia, CT: Gregorc Associates, Inc.

Halasz, F., & Moran, T. P. (1982). Analogy consider harmful. *Proceedings of the CHI'82 Conference on Human Factors in Computer Systems*, 383-386.

Henderson, P. B. & Romero, J. R. (1989). Teaching recursion as a problem-solving tool using standard ML. *SIGCSE Bulletin, 21*(1), 27-31.

Jagodzinski, A. P. (1983). A theoretical basis for the representation of on-line computer systems to naive users. *International Journal of Man-Machine Studies, 18*, 215-252.

Johnson-Laird, P. N. (1980). Mental models in cognitive science. *Cognitive Science, 4*, 71-115.

Johnson-Laird, P. N. (1983). *Mental Models*. Cambridge, MA: Harvard University Press.

Kahney, H. (1983). What do novices know about recursion. *Proceedings of the CHI '83 Conference on Human Factors in Computer Systems*, 235-239.

Karrer, U. (1988). *Comparison of Learning Style Inventories (LSI)*. (ERIC Document Reproduction Service No. ED 296 713)

Katz, N. (1986). Construct validity of Kolb's Learning Style Inventory, using factor analysis and Guttman's smallest space analysis. *Perceptual and Motor Skills, 63*, 1323-1326.

Keefe, J. W. (1987). *Learning Style Theory and Practice*. Reston, VA: NASSP.

Kemeny, J. G., & Kurtz, T. E. (1985). *Back to BASIC: The History, Corruption and Future of the Language*. Reading, MA: Addison-Wesley.

Kessler, C. M., & Anderson, J. R. (1986). Learning flow of control: recursive and iterative procedures. *Human-Computer Interaction. 2*, 135-166.

Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science, 8*, 255-273.

Koffman, E. B. (1992). *Pascal* (4th ed.). Reading, MA: Addison Wesley.

Kolb, D. A. (1976). *Learning Style Inventory: Technical Manual*. Boston, MA: McBer and Company.

Kolb, D. A. (1984). *Experiential Learning*. Englewood Cliffs, NJ: Prentice Hall.

Kolb, D. A. (1985). *Learning Style Inventory*. Boston, MA: McBer and Company.

Kruse, R. L. (1982). On teaching recursion. *SIGCSE Bulletin, 14*(1), 92-96.

Kurland, D. M. & Pea, R. D. (1983). Children's mental models of recursive LOGO programs. *Proceedings of the 5th Annual Conference of the Cognitive Science Society*, Session 4, 1-5.

Lee, P. C., & Mitchell, M. A. (1985). Demystifying LOGO recursion: a storage process model of embedded recursion. *Computers in the Schools, 2*(2,3), 197-208.

Luiten, J., Ames, W., & Ackerson, G. (1980). A meta-analysis of the effects of advance organizers on learning and retention. *American Educational Research Journal, 17*, 211-218.

Marshall, J. C., & Merritt, S. L. (1985). Reliability and construct validity of alternate forms of Learning Style Inventory. *Educational and Psychological Measurement, 45*, 931-937.

Martin, M. R. (1985). Recursion -- a powerful, but often difficult idea. *Computers in the Schools, 2*(2,3), 209-217.

Mayer, R. E. (1979). Can advance organizers influence meaningful learning? *Review of Educational Research, 49*, 371-383.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *Computing Surveys, 13*, 121-141.

Mayer, R. E. (1982). *Diagnosis and Remediation of Computer Programming Skill for Creative Problem Solving. Volume 1: Description of Research Methods and Results. Final Report.* Santa Barbara, CA: University of California. (ERIC Document Reproduction Service No. ED 230 199)

Mayer, R. E. (1985). Learning in complex domains: A cognitive analysis of computer programming. In G. Bower (Ed.), *Psychology of Learning and Motivation*, Vol. 19 (pp. 89-130). New York: Academic Press.

Mayer, R. E. (1987). Cognitive aspects of learning and using a programming language. In J. M. Carroll (Ed.), *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction* (pp. 59-79). Cambridge, MA: MIT press.

Mayer, R. E. (1988). Using conceptual models to teach BASIC computer programming. *Journal of Educational Psychology, 80*(3), 291-298.

McCarthy, B. (1980). *The 4MAT System.* Oak Brook, IL: Excel, Inc.

McCraken, D. D. (1987). Ruminations on computer science curricula. *Communications of the ACM, 30*(1), 3-4.

Messick, S. (1976). *Individuality in Learning.* San Francisco, CA: Jossey-Bass.

Minsky, M. (1975). A framework for representing knowledge. In P. H. Winston (Ed.), *The Psychology of Computer Vision* (pp. 211-277). New York: McGraw-Hill.

Murnane, J. (1991). Models of recursion. *Computers Education, 16*(2), 197-201.

National Association of Secondary School Principals (NASSP). (1979). *Student Learning Styles -- Diagnosing and Prescribing Programs.* Reston, VA: Author.

Newell, A., & Card, S. K. (1985). The prospects for psychological science in human-computer interaction. *Human-Computer Interaction, 1*, 209-242.

Norman, D. A. (1983). Some observation of mental models. In D. Gentner & A. L. Stevens (Eds.), *Mental Models* (pp. 7-14). Hillsdale, NJ: Erlbaum.

Norman, D. A. (1986). Cognitive engineering. In D. A. Norman & S. W. Draper (Eds.), *User Centered System Design: New Perspectives on Human-Computer Interaction* (pp. 31-61). Hillsdale, NJ: Erlbaum.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas.* New York, NY: Basic Books.

Pinto, J. K., & Geiger, M. A.(1991). Changes in learning-style preferences: A prefatory report of longitudinal findings. *Psychological Reports, 68*, 195-201.

Pirolli, P. L. (1986a). *Problem Solving by Analogy and Skill Acquisition in the Domain of Programming.* (Doctoral dissertation, Carnegie-Mellon University, 1985). *Dissertation Abstracts International, 46*, 4048B.

Pirolli, P. L. (1986b). A cognitive model and computer tutor for programming recursion. *Human-Computer Interaction, 2*, 319-355.

Pirolli, P. L., & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology, 39*, 240-272.

Reading-Brown, M. S., & Hayden, R. S. (1989). Learning styles--liberal arts and technical training: What's the difference? *Psychological Reports, 64*, 507-518.

Reiff, J. C. (1992). *What Research Says to the Teacher: Learning Styles.* Washington, DC: National Educational Association.

Rohl, J. S. (1984). *Recursion via Pascal.* New York: Cambridge University Press.

Ruble, T. L., & Stout, D. E. (1990). Reliability, construct validity, and response-set bias of the revised Learning-Style Inventory (LSI-1985). *Educational and Psychological Measurement, 50*(3), 619-629.

Ruble, T. L., & Stout, D. E. (1991). Reliability, classification stability, and response-set bias of the revised Learning-Style Inventory (LSI-1985). *Educational and Psychological Measurement, 51*(2), 481-489.

Rumelhart, D. E. (1980). Schemata: The building blocks of cognition. In R. J. Spiro, B. C. Bruce, & W. F. Brewer (Eds.), *Theoretical Issues in Reading Comprehension* (pp. 33-58). Hillsdale, NJ: Erlbaum.

Rumelhart, D. E., & Norman, D. A. (1981). Analogical processes in learning. In J. R. Anderson (Ed.), *Cognitive Skills and their Acquisition* (pp. 335-359). Hillsdale, NJ: Erlbaum.

Sasse, M.-A. (1991). How to t(r)ap users' mental models. In M. J. Tauber & D. Ackermann (Eds.), *Mental Models and Human-Computer Interaction 2* (pp. 59-79). New York, NY: Elsevier.

Schank, R. C., & Abelson, R. P. (1977). *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Erlbaum.

Sein, M. K. (1988). Conceptual Models in Training Novice Users of Computer Systems: Effectiveness of Abstract vs. Analogical Models and Influence of Individual Differences. (Doctoral dissertation, Indiana Unversity, 1988). *Dissertation Abstracts International*, *49*, 880A.

Sein, M. K., & Bostrom, R. P. (1989). Individual differences and conceptual models in training novices users. *Human-Computer Interaction*, *4*, 197-229.

Sein, M. K., & Bostrom, R. P., & Olfman, L. (1987). Conceptual models in training novice users. In H. J. Bullinger & B. Shackle (Eds.), *Human-Computer Interaction - INTERACT '87* (pp. 861-867). New York, NY: Elsevier.

Sims, R. R., Veres, J. G., & Shake, L. G. (1989). An exploratory examination of the convergence between the Learning Styles Questionnaire and the Learning Style Inventory II. *Educational and Psychological Measurement*, *49*, 227-233.

Smith, D. M., & Kolb, D. A. (1986). *User's Guide for the Learning Style Inventory: A Manual for Teachers and Trainers*. Boston, MA: McBer and Company.

Snow, R. E. (1986). Individual differences and the design of educational programs. *American Psychologist*, *41*(10), 1029-1039.

Tucker, A. B. (Ed.). (1991). A summary of the ACM/IEEE-CS joint curriculum task force report: Computing Curricula 1991. *Communications of the ACM*, *34*(6), 68-84.

Turkle, S. (1984). *The Second Self: Computers and Human Spirit.* New York, NY: Simon & Schuster.

Van der Veer, G. C., & Felt, M. A. M. (1988). Development of mental models of an office system: A field study on an introductory course. In G. C. van der Veer & G. Mulder (Eds.), *Human-Computer Interaction: Psychonomic Aspects* (pp. 251-272). New York: Springer-Verlag.

Van Merrienboer, J. J. G. (1988). Relationship between cognitive learning style and achievement in an introductory computer programming course. *Journal of Research on Computing in Education, 21*, 181-186.

Van Merrienboer, J. J. G. (1990). Instructional strategies for teaching computer programming: Interactions with the cognitive style reflection-impulsivity. *Journal of Research on Computing in Education, 23*(1), 45-52.

Veres, J. G., Sims, R. R., & Locklear, T. S. (1991). Improving the reliability of Kolb's revised learning style inventory. *Educational and Psychological Measurement, 51*(1), 143-151.

Walsh, W. B., & Betz, N. E. (1985). *Tests and Assessment.* Englewood Cliffs, NJ: Prentice-Hall.

West, C. K., Farmer, J. A., & Wolff, P. M. (1991). *Instructional Design: Implications from Cognitive Science.* Englewood Cliffs, NJ: Prentice Hall.

Widenbeck, S. (1988). Learning recursion as a concept and as a programming technique. *SIGCSE Bulletin, 20*(1), 275-278.

Widenbeck, S. (1989). Learning iteration and recursion from examples. *International Journal of Man-Machine Studies, 30*, 1-22.

Wiersma, W., & Jurs, S. G. (1990). *Educational Measurement and Testing* (2nd ed.). Needham Heights, MA: Allyn and Bacon.

Wilson, D. K. (1986). An investigation of the properties of Kolb's Learning Style Inventory. *Leadership and Organization Development Journal, 7*(3), 3-15.

Wittrock, M. C. (1985). Learning science by generating new concepts from old ideas. In L. H. T. West & A. L. Pines (Eds.), *Cognitive Structure and Conceptual Change* (pp. 259-266). Orlando, FL: Academic Press.

Young, R. M. (1983). Surrogates and mappings: two kinds of conceptual models for interactive devices. In D. Gentner & A. L. Stevens (Eds.), *Mental Models* (pp. 35-52). Hillsdale, NJ: Erlbaum.

Zuboff, S. (1988). *In the Age of the Smart Machine.* New York: Basic Books.

Vita

Cheng-Chih Wu was born in TaiChung, Taiwan, R.O.C., on January 15, 1957, the son of Koan Wang Wu and Bor-Yeu Wu. He received a Bachelor of Education in Industrial Education in 1981, and a Master of Education in Industrial Education in 1985, from National Taiwan Normal University, Taiwan, R.O.C. He worked as a lecturer in Department of Electronics at Hwa-Hsia Junior College, Taipei, Taiwan, from 1985-1986. During the following three years, he was employed as a teaching assistant and a lecturer in Department of Computer and Information Education at National Taiwan Normal University. In September 1989, he entered the Graduate School of The University of Texas at Austin to work on a doctoral degree in Computer Science Education. He is married to the beautiful Shih-Ching Wang Wu in 1986 and has one son, Pei-Shin.

Permanent address: 7F, 5 - 8, Ln. 236, Sec. 5, Roosevelt Rd.
Taipei, Taiwan, R.O.C.

This dissertation was typed by the author.