In presenting the dissertation as a partial fulfillment of the requirements for an advanced degree from the Georgia Institute of Technology, I agree that the Library of the Institute shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish from, this dissertation may be granted by the professor under whose direction it was written, or, in his absence, by the Dean of the Graduate Division when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from, or publication of, this dissertation which involves potential financial gain will not be allowed without written permission.

7/25/68
A STUDY OF INFORMATION CONTROL IN
COMPUTER-AIDED INSTRUCTION

A THESIS
Presented to
The Faculty of the Division of Graduate
Studies and Research
By
Margaret Elizabeth Dexter

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
in the School of Information and Computer Science

Georgia Institute of Technology
August, 1972
A STUDY OF INFORMATION CONTROL IN
COMPUTER-AIDED INSTRUCTION

Approved:

Philip J. Siegmann, Chairman

Lucio Chiaravillo

Pranas Pande

Gordon Pask

Date approved by Chairman: August 23, 1972
ACKNOWLEDGMENTS

I would like to thank Dr. Vladimir Slamecka for his assistance and support.

In addition, I would like to thank the members of the reading committee, Dr. Lucio Chiaraviglio and Dr. Pranas Zunde for their constructive criticism and advice. I would like to thank Dr. David E. Rogers for serving on the examining committee. I would also like to thank Dr. William J. Lnenicka for reading and commenting on a draft of the dissertation.

I am grateful to Dr. Gordon Pask for his willingness to review a draft of the dissertation and to serve as a visiting member of the examining committee, and especially for sharing my enthusiasm for this work.

Special thanks go to my thesis advisor, Dr. Philip J. Siegmann, for his suggestions and technical support, for his encouragement, patience and friendship, for the many hours of discussion which gave rise to the research reported herein, and for the prospect of future collaboration as we continue the work begun here.

I also would like to express appreciation to my mother for typing the preliminary drafts and to the students who served as subjects in the experiment.

This research was partially supported under Grant GN-655 from the National Science Foundation. This support is gratefully acknowledged.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF ILLUSTRATIONS</td>
<td>viii</td>
</tr>
<tr>
<td>SUMMARY</td>
<td>x</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>I. COMPUTER-AIDED INSTRUCTION AND THE QUESTION OF LOCUS OF CONTROL.</td>
<td>1</td>
</tr>
<tr>
<td>The Conventional Approach to CAI</td>
<td></td>
</tr>
<tr>
<td>The Question of Locus of Control</td>
<td></td>
</tr>
<tr>
<td>The Need for a Dual-Mode CAI System</td>
<td></td>
</tr>
<tr>
<td>Plan of Presentation</td>
<td></td>
</tr>
<tr>
<td>II. THE CONTROL FUNCTION IN CAI</td>
<td>8</td>
</tr>
<tr>
<td>The Control Hierarchy</td>
<td></td>
</tr>
<tr>
<td>Levels of Memory</td>
<td></td>
</tr>
<tr>
<td>Instructional Processes</td>
<td></td>
</tr>
<tr>
<td>An Example of an Instructional Process</td>
<td></td>
</tr>
<tr>
<td>Communication between Processes</td>
<td></td>
</tr>
<tr>
<td>Topics and Sub-Topics</td>
<td></td>
</tr>
<tr>
<td>Summary</td>
<td></td>
</tr>
<tr>
<td>III. AN ANALYSIS OF THE CONTROL FUNCTION IN OTHER CAI SYSTEMS</td>
<td>29</td>
</tr>
<tr>
<td>The PLATO System</td>
<td></td>
</tr>
<tr>
<td>The Use of ELIZA for CAI</td>
<td></td>
</tr>
<tr>
<td>A Generative CAI System</td>
<td></td>
</tr>
<tr>
<td>The SCHOLAR System</td>
<td></td>
</tr>
<tr>
<td>Discussion</td>
<td></td>
</tr>
<tr>
<td>IV. THE DESIGN OF THE GEORGIA INSTITUTE OF TECHNOLOGY INSTRUCTIONAL TRANSLATOR (GITIT)</td>
<td>44</td>
</tr>
<tr>
<td>The Mockup</td>
<td></td>
</tr>
<tr>
<td>A Modular Approach to Instructional Programs</td>
<td></td>
</tr>
<tr>
<td>Indexing Instructional Modules</td>
<td></td>
</tr>
<tr>
<td>Backward and Forward Pointers</td>
<td></td>
</tr>
<tr>
<td>The Data Structure for GITIT</td>
<td></td>
</tr>
<tr>
<td>The Resulting CAI System</td>
<td></td>
</tr>
<tr>
<td>APPENDIX</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>D. THE COMPUTER PROGRAM FOR GITIT</td>
<td>270</td>
</tr>
<tr>
<td>E. ALGORITHMS FOR GENERATING THE SCRIPT</td>
<td>285</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>292</td>
</tr>
<tr>
<td>VITA</td>
<td>295</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Major Terms Before Run 1</td>
</tr>
<tr>
<td>2.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Major Terms After Run 1</td>
</tr>
<tr>
<td>3.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Minor Terms Before Run 1</td>
</tr>
<tr>
<td>4.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Minor Terms After Run 1</td>
</tr>
<tr>
<td>5.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Major Terms Before Run 2</td>
</tr>
<tr>
<td>6.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Major Terms After Run 2</td>
</tr>
<tr>
<td>7.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Minor Terms Before Run 2</td>
</tr>
<tr>
<td>8.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Minor Terms After Run 2</td>
</tr>
<tr>
<td>9.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Comparison of Terms</td>
</tr>
<tr>
<td>10.</td>
<td>Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Final Definitions of All Major Terms</td>
</tr>
<tr>
<td>11.</td>
<td>Rectitude of Examples</td>
</tr>
<tr>
<td>12.</td>
<td>Rectitude of Recall of Major and Minor Terms</td>
</tr>
<tr>
<td>13.</td>
<td>Order of Processing of Major Terms During Run 1</td>
</tr>
<tr>
<td>14.</td>
<td>Number of Communication Events in Discussion and Tutorial Mode During Run 1</td>
</tr>
<tr>
<td>15.</td>
<td>The Adequacy of Machine Responses in Discussion Mode</td>
</tr>
<tr>
<td>16.</td>
<td>The Command Language Used for Instructional Programs</td>
</tr>
<tr>
<td>Table</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>17. Major Terms for Each Module in Experimental Materials</td>
<td>197</td>
</tr>
<tr>
<td>18. Minor Terms in Experimental Materials</td>
<td>198</td>
</tr>
<tr>
<td>19. Primitive Terms in Experimental Materials</td>
<td>199</td>
</tr>
<tr>
<td>20. Words to be Translated Prior to the Decomposition Process</td>
<td>200</td>
</tr>
<tr>
<td>21. Definitions Given by $S_1$</td>
<td>248</td>
</tr>
<tr>
<td>22. The Record Format for PI Modules</td>
<td>259</td>
</tr>
<tr>
<td>23. Control Commands Recognized by the GITIT System</td>
<td>273</td>
</tr>
</tbody>
</table>
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Student Computer Dialogue</td>
<td>10</td>
</tr>
<tr>
<td>2.</td>
<td>Control of the Student Computer Dialogue</td>
<td>12</td>
</tr>
<tr>
<td>3.</td>
<td>Control Hierarchy in CAI</td>
<td>16</td>
</tr>
<tr>
<td>4.</td>
<td>Memory Systems</td>
<td>17</td>
</tr>
<tr>
<td>5.</td>
<td>Embedding in Instructional Processes</td>
<td>23</td>
</tr>
<tr>
<td>6.</td>
<td>The Decomposition of a Student's Goal into Subgoals</td>
<td>26</td>
</tr>
<tr>
<td>7.</td>
<td>Average Confidence Estimate $\alpha$ and Rectitude $\theta$ for $S_1$</td>
<td>85</td>
</tr>
<tr>
<td>8.</td>
<td>Average Confidence Estimate $\alpha$ and Rectitude $\theta$ for $S_2$</td>
<td>86</td>
</tr>
<tr>
<td>9.</td>
<td>Average Confidence Estimate $\alpha$ and Rectitude $\theta$ for $S_3$</td>
<td>87</td>
</tr>
<tr>
<td>10.</td>
<td>Average Confidence Estimate $\alpha$ and Rectitude $\theta$ for $S_4$</td>
<td>88</td>
</tr>
<tr>
<td>11.</td>
<td>Average Confidence Estimate $\alpha$ and Rectitude $\theta$ for $S_5$</td>
<td>89</td>
</tr>
<tr>
<td>12.</td>
<td>Total Task Uncertainty $H^*$ for all Five Students on Run 1</td>
<td>97</td>
</tr>
<tr>
<td>13.</td>
<td>Graphical Data for $S_1$</td>
<td>99</td>
</tr>
<tr>
<td>14.</td>
<td>Graphical Data for $S_2$</td>
<td>100</td>
</tr>
<tr>
<td>15.</td>
<td>Graphical Data for $S_3$</td>
<td>101</td>
</tr>
<tr>
<td>16.</td>
<td>Graphical Data for $S_4$</td>
<td>102</td>
</tr>
<tr>
<td>17.</td>
<td>Graphical Data for $S_5$</td>
<td>103</td>
</tr>
<tr>
<td>18.</td>
<td>Dependence Matrix for Major Terms Learned During Run 1</td>
<td>106</td>
</tr>
<tr>
<td>19.</td>
<td>Precedence Graph for Major Terms Learned During Run 1</td>
<td>108</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>20. Embedding Structures for all Students on Run 1</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>21. Goal Decomposition Diagram for $S_1$ During Run 1</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>22. Goal Decomposition Diagram for $S_1$ During Run 2</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>23. Goal Decomposition Diagram for $S_1$ During Run 3</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>24. Goal Decomposition Diagram for $S_3$ During Run 1</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>25. Goal Decomposition Diagram for $S_3$ During Run 2</td>
<td>133</td>
<td></td>
</tr>
<tr>
<td>26. The Basic TOTE Unit</td>
<td>142</td>
<td></td>
</tr>
<tr>
<td>27. A Strategy for Learning Definitions</td>
<td>143</td>
<td></td>
</tr>
<tr>
<td>28. A Strategy for Learning Unrelated Items</td>
<td>144</td>
<td></td>
</tr>
<tr>
<td>29. A Strategy for Learning Related Items</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>30. A Strategy for Learning the Relationship Between Two Items</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>31. The Dependence Matrix for Experimental Materials</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>32. The Dependence Graph for Experimental Materials</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>33. The Complete Protocol for $S_1$ on Run 1</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>34. The Complete Protocol for $S_1$ on Run 2</td>
<td>237</td>
<td></td>
</tr>
<tr>
<td>35. The Complete Protocol for $S_1$ on Run 3</td>
<td>245</td>
<td></td>
</tr>
</tbody>
</table>
SUMMARY

The research reported in this dissertation consists of three phases: the development of a qualitative model of the instructional process in Computer-Aided Instruction (CAI), the subsequent design and implementation of an experimental CAI system, and the observation and analysis of student interaction with this system.

CAI is viewed as a dialogue between a student and a computer. In the qualitative model the student and the computer are described as hierarchically organized control systems. The dialogue between the student and the computer is assumed to occur at distinct levels. The dialogue concerns the subject matter, understanding of the subject matter, goals and strategies for attaining those goals, and the choice of a topic area at levels 0, 1, 2 and 3, respectively.

In accordance with the model, an experimental CAI system (GITIT) was designed and implemented on the Burroughs B-5500 with student interaction via a teletype terminal. GITIT provides the capability for either the student or the computer to control at levels 0, 1 and 2. Control commands are provided whereby control may be shifted dynamically between the student and the computer.

GITIT has two modes of operation: discussion mode and tutorial mode. In discussion mode the student may ask any question or make any statement concerning the selected topic. In tutorial mode, GITIT administers a segment of instructional material called a module.
A student may interrupt a module at any time either to place the system in discussion mode or to cause another module to be administered. He may later cause the machine to resume the interrupted module.

The data base for GITIT consists of a collection of short segments of programmed instruction called modules, and a list structure which supports a student computer dialogue concerning the subject content of the modules. A prescription is given for the generation of instructional materials to teach a set of consistent definitions. An algorithm for generating the list structure from a set of instructional modules and a specified technical vocabulary is included.

Observation of student performance on the system confirmed expectations based on prior research as to the variability of individual strategies and effectiveness in a learning task.

GITIT is adequate to support a wide range of learning tasks and is efficient with respect to computer processing time, memory utilization, and the preparation of instructional materials.
CHAPTER I

COMPUTER-AIDED INSTRUCTION AND THE
QUESTION OF LOCUS OF CONTROL

Computer-Aided Instruction (CAI) is here viewed as a dialogue between a student and a computer directed toward a learning goal. The objectives of the research reported here are as follows:

1. To explicate the concept of control of the instructional process in current CAI systems.

2. To design and implement an experimental CAI system with sufficient flexibility to permit the investigation of the control issue in student-system interaction with respect to data structures, query capabilities, and task requirements.

3. To clarify the range of student-system interactions which can be supported by current CAI systems with particular reference to those aspects of control required for adaption to individual differences in prior knowledge or learning strategy.

4. To perform an empirical investigation of some of the effects of permitting the student to exert control over the instructional process.

5. To describe the implications of these findings on future CAI systems in terms of their cost effectiveness.

These objectives have been realized by the development of a qualitative model of the instructional process in Computer-Aided
Instruction, by the design and implementation of an experimental CAI system which conforms to this model, and by the observation and analysis of student interaction with this system.

The Conventional Approach to CAI

Early efforts in the area were based on the premise that an instructor could program the computer in such a way that it could present information to the student, ask the student a question, and then utilize the student's answer to determine the next computer action. In order for this conventional programmed-instruction to be effective, the instructor must anticipate all of the correct and incorrect answers which a student might give, and then specify the appropriate action for each answer.

There are several difficulties with this approach. It can only be expected to work in well-defined subject areas where the questions have "right" and "wrong" answers which can be clearly identified. Even then the preparation of instructional materials is tedious and success depends upon the teacher's ability to anticipate possible student responses and to specify appropriate actions. Despite the emergence of numerous author languages designed to facilitate the development of instructional materials, the effort involved discourages many educators.

A far more serious problem is that the student cannot take the initiative in the instructional process. He must learn the material presented to him in the order selected by the instructor who prepared the lesson, and he cannot ask questions. Although different students
may take different paths through the same material based on their responses to questions, it is nevertheless the teacher rather than the student who prescribes the alternative paths.

Conventional programmed-instruction may be adequate for many purposes. However, there appear to be many teaching situations for which these methods may be inappropriate.

Recent development in CAI indicate a trend toward systems which combine a query or question-answering system with a programmed-instruction system, thus permitting the student to ask questions. This raises many important issues concerning the effect of permitting the student to exert control over the instructional process.

The Question of Locus of Control

An important question in the area of CAI, as in all areas of man-machine communication, is the question of locus of information control. Stated briefly the question is this: Under what conditions should the teacher (or computer) control the information flow, and under what conditions should the information flow be controlled by the student?

It is assumed that the instructional process is a dialogue between the student the computer directed toward a learning goal, and that either the student or the computer may monitor the student's progress toward this goal and control the selection, order and redundancy with which instructional materials are presented. This qualitative and quantitative control of the information flow we shall call information control.
In conventional programmed-instruction the computer, or more correctly, the author of the instructional materials, has information control. That is, he controls the order, selection and redundancy with which information is presented. Some recent efforts have transferred part of the information control to the student by permitting him to ask questions. But little is known about where information control should lie for most effective learning.

The question of information control needs to be considered in relation to a learning goal. CAI is viewed as a dialogue between a student and a computer directed toward a learning goal. Thus in order to exercise effective information control, either the student or the computer must be able to monitor the student's progress toward the established goal. If the student is to control the dialogue, then he must be able to monitor his progress and act accordingly. If the computer is to control the dialogue, then the computer must be able to evaluate the student's progress toward the goal and control the dialogue.

Since information control must be considered in relation to learning goals, then we must also consider who is to control the selection of learning goals and subgoals. Once a goal is selected there is the consequent problem of determining who shall have information control for the attainment of that goal. It appears that the control of goal selection and information control may need to shift dynamically between the student and the computer as subgoals are established and realized. If control of either of the two mentioned types needs to
shift between the two parties, then it becomes necessary for one of the parties to control this shift. Thus a hierarchy of control is generated. The control hierarchy and other concepts of control in CAI are reflected in the qualitative model which we shall present.

The Need for a Dual-Mode CAI System

It would appear that there are circumstances under which the student can best learn by asking questions and thereby controlling the presentation of material, and there are circumstances under which the student needs to be directed. Thus it would seem desirable to develop a CAI system which can operate either in a student-controlled or a teacher-controlled mode. Such a system would facilitate investigation of the locus of control question and might ultimately lead to an improved version of CAI.

A Dual-Mode CAI system should permit the student to either acquire information in a machine-controlled or tutorial mode, or to ask questions in a student-controlled or discussion mode. If the student asks a question which requires a lengthy explanation as an answer, the tutorial mode could be used for the explanation. If questions arise in the student's mind while he is in the tutorial mode, he could shift the dialogue to the discussion mode to ask the questions and then return to the tutorial mode. Thus locus of information control would shift between the student and the teacher (or computer) as required.

There are two major questions which had to be resolved before such a system could be realized:

1. When should the machine shift from one mode to the other?
2. What data structure is required to support a machine which can operate in either mode?

Before either question could be answered, the question of what is meant here by control had to be considered, and an investigation made to determine the effect of control on student performance. Ideally, the machine should shift from one mode to another whenever the student's performance would be improved by doing so.

The question of the required data structure is also related to the locus of control question. Whether the dialogue is teacher (i.e. machine)-controlled or student-controlled, certain information necessary for control must be available. In the teacher-controlled or tutorial mode, the data structure might include descriptions of the students. In the conventional form of programmed-instruction, the author of the instructional materials relies on some description of potential students albeit vague whenever he specifies actions based on student responses. A model of the student exists in the data structure in the trivial sense that if the student is at a given point in the material he is assumed to know all of what went before and nothing of what follows.

The data structure required for the question-answering mode also may include a model of the student. It is often implicitly assumed in discussing question-answering systems that for any question there is a uniquely defined "correct" answer which the computer must be able to determine by examining the question. But this may not be the case, whether the question is addressed to a computer or to a person. The answer which is appropriate depends not only on the question but on the semantic and pragmatic context in which the question is asked.
In order to investigate the effect of locus of control on student performance it was decided that a mockup of a system which could operate in either a tutorial or a discussion mode would provide useful preliminary information. Such a mockup was built and utilized for informal experimentation leading to the design of a dual-mode CAI system which permits analysis of man-machine interaction.

Plan of Presentation

In Chapter II we present a qualitative model of the instructional process; this model discriminates between levels of control, memory, processes and language. In Chapter III we use this model to discuss a number of current CAI systems. Chapter IV describes a mockup of a dual-mode CAI system and experimentation with this mockup leading to the design of the final experimental CAI system (GITIT). This chapter also contains a functional description of this modular, subject-matter independent system. Chapter V contains an analysis of student interaction with the system and a discussion of the range of tasks which GITIT can support. Chapters VI and VII give a detailed description of GITIT and a prescription for generating instructional materials for the system. Chapter VIII contains conclusions and recommendations for future work.
CHAPTER II

THE CONTROL FUNCTION IN CAI

An early application of control concepts to the study of learning was made by Miller, Galanter and Pribram (Miller, 1960) and extended to CAI by Pask (Pask, 1967, 1968, 1970, 1971). Pask also introduced the concept of a hierarchy of control in cognitive systems. In order to clarify the concept of control in CAI, we will view both the student and the computer as hierarchically organized control systems and model a CAI system as a couple of these two systems.

An underlying assumption in the model presented here is that the student is a self-directed learner. By this we mean that the student approaches the system with a learning goal in mind, and that he uses the CAI system as a retrieval system to obtain the instruction he needs to attain his goal.

Pask suggests that the activity within the brain is analogous, in some sense, to the execution of programs in a digital computer. We adopt this analogy, and understand a process to be either a program in execution or to be formed from other processes by concatenation or embedding. Extending the work of Dijkstra (Dijkstra, 1968) and others, to incorporate multiple levels of control, we analyze the interaction between the student and the computer in terms of co-operating sequential processes.
The Control Hierarchy

Viewing CAI as a dialogue between a student and a computer, there is a sense in which the current speaker has control at least to the extent that the dialogue cannot continue until he speaks. This we shall call immediate control. We will assume that immediate control alternates between the student and the computer as one party asks questions or makes statements and the other party responds.

There is also a sense in which either the student or computer exerts control when one party asks a question or makes a statement to which the other must respond. Formally, we define a communication event to be an ordered pair \((S,R)\) where \(S\) is either a statement or a question and \(R\) a response. Under the assumption that the response is appropriate to the statement or question, the first speaker is said to control a communication event.

In the above definition of a communication event, no distinction is made between a statement and a question because they may serve the same function in the dialogue between the student and the computer. For example, a definition of a node may serve as an appropriate response to "What is a node?", "I need to know what a node is." or "Define node." Thus \(S\) is meant to denote the remark made by the first speaker in a communication event without regard to its linguistic form.

The appropriateness of a response \(R\) to a statement or question \(S\) is not a formal matter, but rather it is empirically determined. This is in contrast to studies by Harrah (Harrah, 1963) and Rescher (Rescher, 1966).
Harrah presents a logical model of communication in which the distinction between questions and answers is determined formally. Thus the distinction between the party asking a question and the party answering is determined at the linguistic level. Rescher has studied the logic of commands within a formal system. A command is formally defined at the syntactic level and the execution of a command is an item in the semantics of the theory, thus the distinction between the commander and the commanded is made at the syntactic semantic interface of the formal language.

In the present work, the distinction between the controlling party and the controlled party is made at a functional rather than at a formal level.

The conventional form of CAI in which the computer presents information to the student and then asks a question which the student must answer is a series of communication events controlled by the computer. Taking System A to be the computer and System B to be the student, we have a feedback system as shown in Figure 1.

![Figure 1. Student Computer Dialogue](image-url)
Immediate control lies first with the computer. The computer emits S and immediate control passes to the student. The student emits R thus concluding the communication event and causing immediate control to pass back to the computer for the initiation of the next communication event.

In a question-answering situation the roles of student and computer are reversed. The student poses questions to which the computer responds. Thus, this type of instructional process is a sequence of communication events which the student controls. Taking the student to be System A and the computer to be System B we again have a feedback system as shown in Figure 1.

The dialogue between the student and the computer involves several different levels of language. We shall say that the communication about the subject matter occurs in the language $L^0$. But, not all of the discussion between the student and the computer has to do with the subject matter. The computer may tell the student that his answer is right or wrong or state that a question was not understood. The student may ask for help or request that a question be repeated. This communication is not about the subject matter, but about the statements transacted in those communication events concerning the subject matter. We shall say that this communication occurs in the language $L^1$.

The student-computer interaction can be modeled by a hierarchically organized control system as shown in Figure 2. The boxes $A^0$ and $B^0$ represent the student and computer respectively as they carry on a dialogue (in $L^0$) about the subject matter.
The boxes $A^1$ and $B^1$ represent the student and the computer as they monitor the $L^0$ dialogue and discuss the instructional process (in $L^1$) as required. Both $A^0$ and $B^0$ contain comparators, i.e. devices by which the student and the computer are assumed to monitor their understanding of the current dialogue and to emit messages to $A^1$ and $B^1$, respectively, if the process is not to their satisfaction. This could cause either an $L^1$ statement to the other party or a change in the instructional process. The change in the instructional process is represented in the diagram by a large arrow which we shall call a *transformation operator*.

We assume that both the student and the computer are following a plan or strategy. The machine is following a teaching strategy in presenting information to the student, and the student is following a
learning strategy in acquiring information from the machine. There are
two different types of strategies. There is one type of strategy which
specifies the order of presentation of material. This we shall call a
selection strategy. There is another type of strategy which accepts the
information in whatever order it is presented. This we shall call a
reception strategy. These roughly correspond to selection and reception
strategies as defined by Bruner, Goodnow and Austin (Bruner, 1956).

We will further assume that each strategy is directed toward a
specified learning goal or subgoal, and that progress toward this goal
is monitored by the party following a selection strategy. A computer
strategy and a student strategy are said to be complementary if one is
a selection strategy, the other a reception strategy and both are
directed toward the same learning goal.

In the conventional form of CAI, the machine uses a selection
strategy because it determines the order of presentation of material.
When the student accepts and attends to all information presented he
is using a reception strategy. It may be possible, however, for a
student to select from what is presented to him only that information
in which he is interested. Or he may exert control over the machine by
giving certain "right" or "wrong" answers if he knows what action his
answer will cause the machine to take. In these instances both the
machine and the student are attempting to use selection strategies.

In the question-answering situation the student follows a selec-
tion strategy in determining which question to ask, and the machine fol-
lows a reception strategy in responding to the student's question.
When the machine presents material other than or in addition to that which the student requested, the machine is employing a selection strategy. The student may shift to a reception strategy and accept the information presented, or he may ignore it.

The languages $L^0$ and $L^1$ correspond to levels of control which we shall call level 0 and level 1. Thus, the student (or the computer) is controlling at level 0 when he (or it) is controlling the communication event in $L^0$. The student (or the computer) is controlling at level 1 when he (or it) is following a selection strategy. We shall call control at the $L^1$ level information control and say that locus of information control is with the student (or computer) when he (or it) is following a selection strategy and thus controlling at level 1.

A third level of control which we shall call procedural control is required to control the choice of goal and strategy, and a third language, $L^2$, is required to discuss goals and strategies. Either the student or the computer may control at level 2. Note that the selection of either a selection or a reception strategy determines locus of information control. Thus control at level 2, i.e. procedural control, is control of control and shall be written control.$^2$

It is assumed that the discussion in $L^2$ is restricted to a set of learning goals within a specified topic area. A fourth level of control and a fourth language, $L^3$, are required to discuss and select a topic area and to determine locus of control.$^2$ This we shall call control of the topic area and we shall say that locus of control$^3$ is with the party who controls at the third level. Under the assumption that the student
is a self-directed learner, we will assume that the student rather than the computer controls the choice of topic area; thus the student is assumed to control at level 3.

A diagram of the resulting hierarchically organized control system is shown in Figure 3. It may be possible to identify additional levels of control, but the four levels shown are adequate for the present purpose.

Levels of Memory

The actions at each level are taken to be the result of executing programs stored in the memory of the student and of the computer. This is consistent with the model of cognitive behavior presented by Pask (Pask, 1968) and with the actual operation of a computer. Both the student and the computer have several different levels of memory. We will assume that each has three levels of memory: long term memory, working memory, and immediate memory. A diagram of the memory systems is shown in Figure 4.

The three levels of memory may be reflected in the computer by the use of physically distinct devices such as tape for long term memory, disk for working memory and core for immediate memory, or the same physical device may be used for functionally distinct levels of memory.

The exertion of control at the $i^{th}$ level (concretely by the computer and by analogy by the student) is achieved by the execution of $i^{th}$ level programs. That is to say, at the $0^{th}$ level, the actions of the computer (and by analogy the student) result from the execution of level
Figure 3. Control Hierarchy in CAI
0 programs within the processing unit of the computer (the brain of the student). Actions at the first level result from the execution of level 1 programs and similarly for level 2 and level 3 programs.

![Figure 4. Memory Systems](image)

During the execution of the level 1 programs, either the student or the computer (depending upon locus of information control) will initiate the execution of a level 0 program which results in an L^0 communication event. It is usually but not necessarily the case that the party with information control controls the resulting communication event.

Subsequent level 0 programs are initiated by the level 1 programs until either the student or the computer exerts control at level 2 to initiate different level 1 programs. This may occur because a level 1 program is completed and the associated goal or subgoal attained,
because it is determined that the goal cannot be reached using that
strategy and the level 1 program terminated or because the level 1
program is temporarily interrupted to permit execution of another level
1 program.

All level 0 and level 1 programs within the selected topic area
are assumed to reside in working memory and to be transferred to immediate memory as they are required for execution.

**Instructional Processes**

The control hierarchy presented in the previous sections should
not be confused with the hierarchy of goals and subgoals. We assume
that the student is a self-directed learner who approaches the system
with a learning goal or objective. It may be the case that this goal
was anticipated by the instructor who prepared the instructional
materials, in which case the machine may have a specified strategy for
attaining the desired goal. But if this is not the case, the student
may be able to decompose his goal into subgoals for which the machine
does have strategies, and then utilize these strategies to construct a
strategy which at least partially satisfies his goal.

In order to study the way in which goals and subgoals are pur-
sued, we will view the student and the computer (or more specifically
the brain of the student and the central processing unit of the com-
puter) as two processors each of which execute programs. Each processor
is assumed capable of multi-programming and in particular is assumed to
be executing programs at levels 0, 1, 2 and 3. Thus the CAI system
composed of a student coupled to a computer is analogous to a computer
system with two physical processors (the CPU of the computer and the brain of the student) each with multiple pseudo-processors. The interaction between the student and the computer can thus be viewed as cooperation between sequential processes in the sense of Dijkstra (Dijkstra, 1968) and others who have studied complex computer operating systems.

An elementary process is here defined to be a program in execution by a pseudo-processor. A process is either an elementary process or it is formed from other processes by concatenation or embedding. An elementary process resulting from the execution of a level 0 program will be called an elementary level 0 process and similarly for elementary level 1, 2 and 3 processes.

When a student approaches the system with a learning goal in mind, he is assumed to execute a level 3 program and the machine executes a level 3 program, both of which yield a level 3 process, $P^3$. These processes serve to select the topic area. We will view the entire interaction between the student and the computer as a level 3 process within which are embedded level 2, 1 and 0 processes.

Having selected an appropriate topic area for his goal, both the student and the machine begin to execute level 2 programs in order to select the first subgoal and appropriate strategies. Thus a level 2 process is embedded in the level 3 process. Having selected the first learning goal and strategies for attaining this goal, the appropriate level 1 processes are begun. These processes in turn cause the execution of level 0 programs which produce level 0 processes. At the
completion of the level 1 process another level 1 process may be initiated to pursue another learning goal, thus concatenating processes.

As stated previously, we assume that the student is a self-directed learner and that he controls the choice of a topic area. Thus the level 3 processes are student controlled. We also assume that the student initially exerts control at the second level to establish a learning goal, but that the student may transfer locus of control to the machine. Then the machine may recognize the need for a subgoal and exert control at the second level thereby initiating level 1 processes. Therefore, we will assume that the level 2 programs are dual-mode in the sense that either the student or the computer may be in control at the second level.

The level 1 programs correspond to strategies which are either selection strategies or reception strategies, thus a level 1 process is either under student control or computer control. A level 1 process which results from a selection strategy will be called a controlling level 1 process and a process which results from a reception strategy will be called a non-controlling level 1 process.

A controlling level 1 process determines the order of presentation of instructional material and monitors the student's progress toward the established goal or subgoal. When the goal is reached, the controlling level 1 process indicates to the level 2 process in which it is embedded that the goal has been attained and the level 1 processes are terminated. Either the controlling or the non-controlling level 1 process may indicate to the level 2 process in which it is embedded
that the process should be prematurely terminated.

Upon the termination of any process, the process at the next higher level of embedding is resumed. This process is then active until it is terminated or temporarily interrupted for the embedding of a new process.

Processes may be combined by concatenation or embedding to produce new processes. Interleaving of processes has not been considered. That is, if process X initiates process Y then process Y must be terminated before process X is resumed, and it is not possible to later continue process Y. However, another process resulting from the execution of the same program as X may later occur, and may even be embedded in process X.

An Example of an Instructional Process

In the following example we use $P^0$, $P^1$, $P^2$ and $P^3$ to denote processes at levels 0, 1, 2 and 3, respectively. Subscripts are used to distinguish between multiple processes at the same level.

Assume that the student wishes to learn the difference between a stack and a queue. He initiates process $P^3_1$ in which he and the machine engage in an $L^3$ dialogue until they determine a topic area which includes stacks and queues. Assume that this topic area is called "linear data structures." Having selected this area, both the student and the machine place the necessary programs in intermediate storage and begin executing their level 2 programs, process $P^2_1$. They must then agree on a first subgoal.

We will assume that the computer has a selection strategy for teaching the definition of a "stack," and the student agrees to use a
reception strategy to learn this definition. Processes $P^1_1$ in which the computer has information control and in which the student uses a reception strategy are initiated by the $P^2_1$ processes. The computer then initiates process $P^0_1$ in which the student is presented the definition of a stack. (See Figure 5.)

Summarizing the above actions, we have

**Level 3:** Topic selected and processes $P^2_1$ initiated

**Level 2:** First subgoal selected and processes $P^1_1$ initiated

**Level 1:** Processes $P^0_1$ initiated

**Level 0:** Student presented a definition

Thus we have a level 0 process embedded in a level 1 process embedded in a level 2 process which is in turn embedded in a level 3 process for both the student and the computer. The computer has just presented an item of information and is awaiting a response from the student. The student is reading and attempting to understand the definition. Level 0 processes are active; level 3, 2 and 1 processes are suspended. An $L^0$ communication event controlled by the computer is incomplete. The student has immediate control.

The student may now do any of several different things. He may give an $L^0$ response thus completing the $L^0$ communication event, or he may initiate another communication event in a higher level language. He may ask for help at the first level; he may alter the goal or strategy at the second level; or he may return to the third level and change topic areas.

Let us assume that he sees no reason to change topics and he does not wish to reject the present strategy, but that he does not
Figure 5. Embedding in Instructional Processes
understand the definition because he lacks a prerequisite concept. He cannot control the order of presentation within the current level 1 process, but he can recursively call his \( L^2 \) program thus initiating process \( P^2_2 \) within which he can initiate another level 1 process, \( P^1_2 \). So he returns to the second level of control and initiates process \( P^1_2 \), then initiates a new level 1 process, \( P^1_2 \), such that he has information control and the goal is to learn the definition of "linear list" which occurred in the definition of "stack." Having obtained this definition he returns to the suspended process \( P^1_1 \) concerning "stack." We assume that three additional level 0 processes, \( P^0_3 \), \( P^0_4 \) and \( P^0_5 \), are required to attain the goal thus completing the process \( P^1_1 \). He then returns to process \( P^2_1 \) where a third level 1 process, \( P^1_3 \), is initiated to teach the definition of "queue." Upon completion of this process, which involves processes \( P^0_6 \), \( P^0_7 \) and \( P^0_8 \), the student is able to analyze the definitions of "stack" and "queue" to determine the difference without communication with a concurrent process in the computer.

Figure 5 shows a diagram of the processes described above. Although this diagram reflects the decomposition of machine processes into subprocesses, it does not adequately reflect the student processes at the first level. The student has a goal of learning the difference between a stack and a queue which is not reflected in a machine process. A more useful diagram is shown in Figure 6.

The student's learning goal, \( G \), was to learn the difference between a stack and a queue. The computer did not have a predetermined strategy for this goal. Therefore, the student decomposed his goal.
G: DIFFERENCE BETWEEN STACKS AND QUEUES

G₁: STACK (P₁)

G₁₁: LINEAR LIST (P₂)

G₂: QUEUE (P₃)

G₃: COMPARISON STACKS AND QUEUES

Figure 6. Decomposition of Student's Goals into Subgoals
into three subgoals: $G_1$, learning the definition of "stack," $G_2$, learning the definition of "queue," and $G_3$, comparing stacks and queues. The machine had selection strategies for goals $G_1$ and $G_2$. The student was thus able to attain his goal by causing the machine to follow first one strategy and then the other, and then comparing stacks and queues.

While exercising the strategy for $G_1$, the student discovered that he needed to learn what a linear list was. This gave rise to goal $G_{1,1}$ which became a subgoal for $G_1$. Again, the machine had an appropriate strategy which the student caused the machine to embed in the strategy for $G_1$. The student then compared stacks with queues thus attaining $G_3$.

The decomposition of goals into subgoals is shown in Figure 6. The corresponding processes are shown in parentheses beside each goal. Note that there is no machine process corresponding to $G$ because the machine did not have a strategy for teaching the difference between a stack and a queue, nor is there a process corresponding to the comparison of stacks and queues.

Communication Between Processes

Two types of communication occur between processes: communication between processes of different levels within the same subsystem (i.e. either the student or the computer), and communication between processes at the same level in different subsystems. These paths of communication can be seen in the diagram in Figure 3.
The communication within the same subsystem is assumed to occur only between adjacent levels of control. The transformation operator represented by the large arrow serves to initiate and modify processes at the next lower level. Results from a comparator within a process signal to the controlling process at the level above.

It is not at all clear what causes such a signal within the human processing. (What causes you to abandon one method of solving a problem and try another?) But it is clear that thinking does occur at various levels, i.e. we monitor ourselves solving a problem, and that some form of communication does occur between the processes at various levels. Within the computer similar communication occurs between various levels of control.

The dialogue between the student and the computer results from communication between processes at the same level of control, the dialogue in the languages $L^0$, $L^1$, $L^2$ and $L^3$ being communication between processes at levels 0, 1, 2 and 3.

**Topics and Sub-Topics**

Two hierarchies have been considered: a hierarchy of control and a hierarchy of goals and subgoals. There are other hierarchies as well.

We assume that a topic area is selected and that the dialogue is conducted within this area. But a topic may have sub-topics and may itself be sub-topic of another area. We assume that the shift to another topic or to a more specific or general topic constitutes a change of the $L^0$ language as well as a change in goals and strategies.
It also appears that examples involve a different $L^0$ language and that some topic areas may serve as examples for other more abstract topics thus producing a hierarchy of abstraction.

There may also be a language meta-language hierarchy within an $L^0$ language. For example, the $L^0$ language for teaching algebra may include natural language and algebraic expressions. This should not be confused with the hierarchy of languages corresponding to the control hierarchy.

**Summary**

The student and the computer are viewed as hierarchically organized control systems which interact at each of four levels of control. The actions of the student and the computer at each of these levels are taken to be the execution of programs stored within their memory. Elementary processes, which are programs in execution, may be combined to form new processes by concatenation and embedding. These processes communicate with each other, communication being permitted between processes at adjacent levels within the student or the computer and between processes at the same level one within the student and the other within the computer.
CHAPTER III

THE CONTROL FUNCTION IN PREVIOUS CAI SYSTEMS

In this chapter we survey related work in CAI and analyze it in light of the control function presented in the previous chapter. This permits a clarification of what other workers have meant by control and a comparison of different systems as regards the control function.

The PLATO System

The PLATO system developed at the University of Illinois is one of the most extensive CAI systems in operation. This system began in 1954 with one student-terminal connected to the ILLIAC I. The version considered here, known as PLATO III, consists of 20 student terminals attached to a CDC 1604 computer (Avner, 1969, Bitzer, 1967, 1970).

The student terminal consists of a keyboard and a TV screen. The keyboard contains the usual alphabetic and numeric characters, a variety of special characters, and a number of special purpose keys. All student input is through this keyboard. The TV screen is used to display messages from the computer to the student including diagrams and slides as well as verbal material. We will consider that the languages $L^0$, $L^1$, etc. may include anything displayed on the TV screen as well as anything entered through the keyboard.
Only two levels of memory are used by the PLATO system. Lessons reside on disk (long-term memory). When a student enters a lesson, the entire lesson is transferred to core storage (immediate memory) where it remains so long as the student is within that lesson. Thus the activity at level 3 causes the transfer of a lesson to immediate memory. A lesson corresponds to a collection of level 1 programs.

It is not clear from the available reports how a topic area or lesson is selected. We will assume that the student enters a lesson number or indicates the desired lesson in some other manner. There is no indication of dialogue concerning topic areas.

Each lesson is prepared using a particular teaching rule or logic. A logic corresponds to a level 2 program for the computer. A number of different logics have been developed for the PLATO system. The tutorial logic and the inquiry logic are the most frequently used logics and the only ones which will be considered here.

The Tutorial Logic

In a typical program using the tutorial logic, the keys on the keyboard are divided into two types: those used by the student to enter responses to questions and those used by the student to control his progress through the instructional material. These correspond to $L^0$ and $L^1$ statements, respectively.

The lesson material is composed of (1) a main sequence of materials which every student must use, (2) help sequences provided for students who have difficulty, and (3) supplementary material for students who want to do additional or more challenging work or for
students who need remedial work.

The student is presented basic information and asked a question. He uses the keyboard to enter his response. This constitutes a machine-controlled $L^0$ communication event.

If the question is too difficult he may press the "help" key (an $L^1$ statement) which causes the machine to select an appropriate help sequence. Although the student requests help, it is the computer rather than the student who determines the help sequence to be administered. Thus we say that the computer exerts control at level 2. An incorrect answer may also cause the computer to activate a help sequence. Upon completion of the help sequence the main sequence is resumed.

Other special purpose keys ($L^1$ statements) are used by the student to ask for the answer to a question, to indicate that he is ready to continue, to request supplementary material, etc.

We have a level 2 process corresponding to the tutorial logic. This process initiates a level 1 process corresponding to the main sequence of material within a lesson. Locus of information control is with the computer. The level 1 process in turn initiates level 0 processes corresponding to specific units of material.

If the student requests help or the computer determines that a help sequence is indicated, a level 2 process is embedded in the level 1 process corresponding to the main sequence. This level 2 process initiates a level 1 process corresponding to the help sequence. Upon completion of this level 1 process the level 1 process corresponding to
the main sequence is resumed. The supplementary material is handled in a similar manner.

Locus of information control is with the computer whenever the tutorial logic is used because all sequences assume that the computer rather than the student determines the order of presentation of material. Locus of control\(^2\) is also with the computer because it is the computer rather than the student who selects the help sequences and supplementary materials.

The Inquiry Logic

In a typical program using the inquiry logic, the keys are divided into three types: those used for entering responses, those used as control keys for particular problems, and those used to search for particular information. These correspond to \(L^0\), \(L^1\), and \(L^2\) statements, respectively.

The lesson material is divided into two categories: (1) a set of problems, and (2) reference material which the student may use in solving the problem. The student begins by familiarizing himself with the available information. This information may include definitions, theorems, examples, a table of Laplace transforms or anything else the lesson author thinks he might need. He may obtain any of this information at any time.

The problems are presented to the student together with text and questions much as in the tutorial logic. Locus of information control is with the computer. The student may, however, exert control at the second level to obtain the information he needs from the reference
material. Thus locus of control is with the student.

The inquiry logic is represented within the computer by a level 2 program. The student controls at level 2. Execution of the level 2 program by the computer is thus a non-controlling second level process.

The student explores the available information by initiating level 1 processes to see what information is available. There does not appear to be any real dialogue about goals and strategies but rather a trial initiation of various level 1 processes.

When the student is ready for a problem he initiates a level 1 process in which locus of information control is with the computer. He then returns to the second level and initiates additional level 1 processes as required until he can solve the problem.

Comparison

Bitzer (Bitzer, 1967) stresses that there is a difference between a request for a particular kind of information in the inquiry logic and a request for help in the tutorial logic. In the inquiry approach, the student controls the sequence in which he obtains special types of information. In the tutorial approach he gets preprogrammed help based on the type of difficulty he is having. We reflect this by saying that locus of control is with the student in the inquiry logic but with the computer in the tutorial logic. Locus of information control is with the computer in both instances.

In an experimental study, a course teaching circuit analysis was programmed and taught in both the inquiry and tutorial logic. Students
taught by the inquiry logic showed greater problem-solving ability than students taught by the tutorial method (Bitzer, 1967).

The Use of ELIZA for CAI

The ELIZA program which was initially developed by Weizenbaum (Weizenbaum, 1966) to study natural language communication between men and computers, has been applied to CAI by Taylor (Taylor, 1968). The student is given a problem, and must ask questions to obtain the information required to solve the problem. Then he must explain his solution to the computer. The computer can also ask questions, if necessary, in order to "understand" what the student is saying.

ELIZA was not designed as a question-answering system, and its initial application was the simulation of a psychiatrist because this was a communication situation in which one party to the conversation, the psychiatrist, was not required to exhibit knowledge of the external world. In the initial version of ELIZA, an attempt was made to conceal a lack of understanding on the part of the machine while producing an illusion of natural language conversation. In a later version of ELIZA, Weizenbaum (Weizenbaum, 1967) has attempted to program the machine in such a way that it will reveal its lack of understanding, and ask for clarification if necessary.

Both versions of the ELIZA program utilize a script which contains the information required by the computer to support a dialogue with the user.

The second version of ELIZA differs from the first in two significant ways. First, it contains an evaluator which is a computer
program capable of controlling at what we have called the second level. Second, it uses local and global scripts where a local script supports a dialogue within a sub-context of the semantic context of the global script. A global script together with the associated local scripts and their subscripts thus correspond to a hierarchy of generalization and specialization with the global script at the apex. This should not be confused with the hierarchy of control.

The choice of a topic area would involve the selection of an evaluator and a global script with its associated subscripts. The evaluator is a level 2 program and each script a level 1 program. (An alternative analysis would take each script or subscript to represent a topic area with a single level 1 program.) There is no indication that the choice of a topic area or any dialogue about topic areas is a part of the system.

The initial dialogue between the student and the computer is supported by a global script. Subscripts dealing with more specific contexts are called as needed. Subscripts can themselves have subscripts to arbitrary depths with the flow of the conversation from general to specific and back to general again accomplished by using different scripts at each level of generalization.

Weizenbaum (Weizenbaum, 1967) gives two examples of dialogues between the student and the computer. These are supported by two different scripts with two different evaluators. In one case locus of information control is with the student and the computer performs computations specified by the student. The other example uses a script
prepared by Taylor as a CAI application in which locus of information control shifts from computer to student and then to the computer again as the student solves a problem.

The interesting example for our purposes is the CAI application. The student is presented a problem which he must solve. He is given an opportunity to ask questions. When he is ready to give a solution to the problem he types "ready" and the computer quizzes him about his solution.

Locus of information control is with the computer when the problem is presented. The computer then exerts control at the second level to shift information control to the student so that he can ask questions. When the student indicates that he is ready to suggest a solution, information control shifts to the computer and the student is questioned about his solution. Thus this script corresponds to a dual-mode level 1 program in which locus of information control can be either with the student or the computer. Locus of control is with the computer.

An alternative analysis would be that locus of information control actually remains with the computer throughout the entire dialogue, the student controlling only the L communication events because he is actually permitted very little control over the presentation, that is, he must ask the right questions. But he must operate under a selection strategy to formulate the questions, thus we say that he has locus of information control.
The important aspect of this effort in relation to the present research is the recognition of the importance of having scripts which correspond to various contexts and subcontexts.

**A Generative CAI System**

Wexler (Wexler, 1970) has developed a CAI system which uses an information network to generate questions and evaluate responses. There are three modes of operation: teacher, student and dialogue. The teacher mode, which is used by instructors to enter instructional materials, will not be considered here.

The student progresses through the actual lesson material in student mode. The information presented and questions asked are based on a strategy prepared by the instructor, thus locus of information control is with the computer.

The lesson material may be a detailed sequence of statements, questions, remedial material, etc., or it may be taken from an information net prepared by the instructor and used by the computer to generate an instructional sequence. The student may request a preprogrammed help sequence if one has been provided, and he may be permitted to enter the dialogue mode to ask questions.

The student first selects a lesson by entering an identification number, thus the student controls choice of topic area. There is no dialogue other than the request for a lesson whose number the student must obtain before approaching the system.

The computer has two level 1 programs for each lesson, one of which is executed in student mode and one which is executed in dialogue
mode. The student mode level 1 program is executed first. This program may be interrupted to execute the dialogue mode level 1 program.

Locus of information control is with the student in dialogue mode and with the computer in student mode. The student may request a shift to dialogue mode from student mode ($L^2$ communication) but the shift of mode (level 2 control) must be instigated by the computer. Thus locus of control is with the computer.

Wexler's objective was to develop an automatic means of generating instructional materials. An automatic strategy is available whereby the machine generates statements and questions from the information net. The strategy amounts to generating a question which the computer can answer. If the information needed to answer the question has previously been presented the computer asks the question, otherwise it presents the information and saves the question for later presentation.

There are several problems with this approach. There seems to be little reason to expect that the order of presentation with such a strategy will be conducive to learning. Preparation of an information net is difficult and time consuming. And response time to a student question is slow, as much as two minutes in one example which was given. It would appear that the latter difficulty is at least partially the result of failing to consider the way in which various levels of memory should be utilized.

The application shown was essentially fact retrieval such as capitals of states or countries, location of cities, population, etc.
Because the information net is based on named relations such as "is the capital of" it does not appear that information other than facts could be accommodated.

The SCHOLAR System

The SCHOLAR system developed by Carbonell (Carbonell, 1970) uses a semantic network (Quillian, 1969) to generate statements and answer questions. Quoting Feigenbaum, Wexler states that the goal of artificial intelligence is "to construct computer programs which exhibit behavior that we call 'intelligent behavior' when we observe it in human beings" (Feigenbaum, 1963) and that his SCHOLAR system is an effort in that direction. He is thus concerned with whether or not the program "knows" what it is talking about. There is no indication that he is concerned with whether or not the student appears intelligent or knows what he is talking about.

The resulting system is able to present factual information and answer factual questions such as "What is the capital of Brazil?" There is no indication that the computer follows any strategy in controlling the flow of information, it just says what it knows and asks questions which it can answer.

In the initial dialogue the student is asked to specify the subject matter to be discussed. He also specifies a number of parameters such as the number of questions, duration of interaction, etc.

The student then specifies the type of interaction, TEST, Q/A or MIXINIT. The computer is considered to have three level 1 programs corresponding to these three types of interaction. It appears that
there is no meaningful control exerted by the computer at the first level in any of these types of interaction, and the distinction between them has to do with the control of communication events at the zero level of control. In the TEST mode the computer controls the $L^0$ communication events and in the Q/A mode the computer controls the $L^0$ communication events. In the MIXINIT mode the student has the prerogative of initiating a new $L^0$ communication event rather than responding to the computer-controlled event. He must later respond.

Carbonell is concerned with the important question of control in relation to man-machine interaction, and has demonstrated the feasibility of a dialogue in which either the student or the machine can take the initiative at the zero level. But he has ignored the question of a learning goal. He states that he takes the semantic network to be a model of the "ideal student." But if that is so, then his ideal student is always assumed to know everything which the machine could teach. What then is the goal and how can progress toward the goal be monitored?

SCHOLAR is an interesting development in that it does permit the initiative to shift dynamically between the computer and the user. But it does not appear that the computer exerts control toward any purpose, or that consideration has been given to any aspect of control other than the control of an $L^0$ communication event.

**Discussion**

None of the systems considered exhibits all levels of control or dialogue in all levels of language as presented in the model.

Although Carbonell specifically addresses himself to the study of control in man-machine interaction, there is no indication that he
has considered control other than at the zero level. The student and the computer share the initiative in $L^0$ communication events. But he does not concern himself with goals and subgoals or the control of the order of presentation of materials which we call information control. Consequently he is unaware of the problem of controlling the choice of goal and the locus of control in pursuing a goal, i.e. locus of control. Topic areas are merely specified by the student with no consideration for topics and sub-topics.

Carbonell's stated objective was to so program a computer that it would exhibit behavior which would, if encountered in humans, be considered intelligent. He may have produced an illusion of intelligent behavior. But it does not appear that Carbonell's efforts contribute to an understanding of the control issue in man-machine interaction in general or in CAI in particular.

Wexler did consider control at level 1, i.e. information control. He provided a mode in which locus of information control is with the student and a mode in which locus of information control is with the computer. He also provided an $L^1$ language in which the student may request help and an $L^2$ language in which the student may request a shift of mode. However, the student is not actually able to cause a shift of locus of information control, thus locus of control is always with the computer. The control of topic area is not considered.

Wexler's primary objective was to facilitate the preparation of instructional materials. It does not appear that he has succeeded. The preparation of an information net is obviously very time consuming. The
program is also very slow in execution largely because he has not given consideration to the effect of levels of memory on the data structure or to the different requirements for a data structure for different modes of interaction.

The PLATO system is difficult to evaluate because it appears that a logic could be prepared which could make it perform in almost any desired way. However, it does not appear that any attention has been given to what might be desirable in light of the control issue in man-machine interaction.

Locus of information control was with the computer in both of the logics considered although it may be the case that logics have been prepared or could be prepared in which locus of information control was with the student. Locus of control was with the student in one logic and with the computer in the other.

It appears that the PLATO system is so flexible that it could do virtually anything. It does not appear that it has been used in ways which contribute to an understanding of the effects of control on CAI.

The second version of the ELIZA program which was used by Taylor displays three levels of control. Locus of control can be either with the student or the computer. The control of the communication event in \( L^0 \) is with either the student or the computer, depending upon the locus of information control. The shift of locus of information control is instigated by the computer; thus locus of control is with the computer.

The important contribution of this effort is the recognition of the importance of context in supporting a man-machine dialogue. The difficulty is that the script is difficult to prepare and there is no
prescribed method for preparing a script. It is also necessary to write an evaluator for each topic area. Because the information required to control the continuity of the conversation and to determine the appropriate action is not contained in the script but in the evaluator, the lines of demarkation between the first and second levels of control are somewhat blurred.

It does not appear that any of the previous approaches provide an adequate framework for studying the effects of control on CAI.
CHAPTER IV

THE DESIGN OF THE GEORGIA INSTITUTE OF TECHNOLOGY INSTRUCTIONAL TRANSLATOR (GITIT)

An experimental CAI system known as the Georgia Institute of Technology Instructional Translator (GITIT) has been designed and implemented. The design process included the development of a mockup and informal experimentation using the mockup. A discussion of this preliminary experimentation is given here in order to explain the rationale for design decisions. A functional description of the resulting CAI system is also given in this chapter. Experimentation with the resulting system is discussed in Chapter V, and a detailed description of the GITIT system is presented in Chapter VI.

The Mockup

A mockup of a CAI system with three levels of control was designed and implemented on the Burroughs B-5500 with user interaction via a teletype terminal. The mockup, which appears to the user to be a CAI system, is a combination of a conversation routine and a routine which administers a programmed-instruction course. When the conversational routine is active we say that the mockup is in discussion mode; when the routine which administers a programmed-instruction course is active we say that the mockup is in tutorial mode.

Locus of information control and control of the L^0 communication events is with the student when the mockup is in discussion mode and
with the computer when the mockup is in tutorial mode. Locus of control is with the student in all but one instance. There is no provision for control of the topic area.

The routine for the discussion mode is an adaption of the first version of the ELIZA program which was initially programmed for the B-5500 by Fricks (Fricks, 1970). The data structure is a script as used in the ELIZA program (Weizenbaum, 1966).

In the discussion mode, order of presentation of material is controlled by the student. He enters a question or a statement and the computer responds. The computer does not retain any information containing previous student inputs, therefore each response is dependent only upon a single input.

The routine for the tutorial mode is an interpreter for an instructional program written in a subset of Coursewriter I, a language developed by IBM. (See Meadow (Meadow, 1970) for a description of this subset of Coursewriter I.)

The data structure for the tutorial mode is a collection of instructional programs each with a unique name. The student is presented with a few lines of information and then asked a question which he must answer. His answer is compared with specified right and wrong answers. If his answer is judged to be correct he is presented with new material, otherwise he may either be given a hint and told to try again or presented remedial material depending upon the instructional program. Thus the order of presentation of material is determined by the computer under control of the instructional program.
Special commands (L^2 statements) are provided which permit the student to shift the mockup from one mode to the other. Whenever he shifts the mockup to tutorial mode he must supply the name of the PI program. A special PI program is provided to supply the student with a list of available instructional programs. The only time the computer exerts level 2 control is to return from tutorial mode at the conclusion of an instructional program.

Embedding to arbitrary depths is permitted. A student may suspend the presentation of one instructional program either to enter discussion mode or to select another instructional program which may in turn be suspended. There is no fixed limit to the number of instructional programs which may be suspended at a given time. Upon completion of an instructional program the program in which it is embedded is resumed at the point of interruption. The computer serves to mark the place in a suspended program.

In terms of the conceptual model, the execution of an instructional program and a dialogue in discussion mode are level 1 processes. A level 2 process is required to call the discussion mode again or to select a new instructional program. These processes may be embedded to arbitrary depth.

Preliminary experimentation with the mockup utilized a diverse collection of instructional programs and a script prepared for another purpose. But the experimentation did permit a demonstration of the ability of the subject to control at the second level and a test of the adequacy of the L^2 commands provided for this purpose.
The interesting result of this experimentation was that subjects were unable to remember the embedding structure. Upon return to a suspended process they often did not know the mode. In one instance, a subject (1) entered discussion mode, (2) requested a special instruction program which explains L2 commands, (3) requested another instructional program to give information on available instructional programs, and (4) finally selected an instructional program on a given topic. After a few communication events he decided to reject that instructional program. At this point he turned to the experimenter in a state of confusion. He was completely unable to recall what he had done, and the only way he could proceed with the experiment was to reinitiate the program.

The preliminary experimentation led to several modifications in the mockup.

In the initial version of the mockup, the computer printed a colon whenever it was ready for a student entry. This was changed to either a D or a T followed by a colon to indicate whether the machine was in the discussion or the tutorial mode.

An L1 command "repeat" was added. This command, which is meaningful only in the tutorial mode, causes the computer to repeat the preceding statement or question in the current level 1 process. This is especially useful when embedding has occurred and the student has returned to a tutorial mode process and cannot remember what he was doing.

These modifications greatly reduce the confusion on the part of students using the mockup. But these results are interesting in that
they indicate a limited human capability for embedding processes. This suggests that it is not necessary to provide a computer program capable of embedding to arbitrary depths. A limit could be placed on the depths of embedding which exceeds human capacity yet is well within the memory limitations of a conventional computer.

A Modular Approach to Instructional Programs

The original design concept called for a collection of instructional programs or lessons each of which had an associated script. It was expected that the student would proceed through a lesson entering discussion mode to ask questions and then return to resume the lesson as needed. It was anticipated that lessons would be written in the conventional manner such that the computer retained information control over the entire lesson except when interrupted by the student for questions.

The difficulty with this approach, in addition to the usual difficulties of preparing instructional materials, was that a student who obtained information in discussion mode might then be led through the same material when he returned to tutorial mode. And a student with partial knowledge of an area or with a specific question in mind would be forced to go through material he did not need in order to locate the specific material he needed.

Consideration of these problems led to a modular approach to the preparation of instructional programs.

Short instructional programs or modules were prepared. The student could then request very specific units of material in tutorial
mode and avoid covering material he already knew. This also greatly simplified the preparation of instructional materials because the instructor could write individual modules without concern for all possible paths through the modules.

In the process of preparing and testing the experimental materials, it became clear that the student should be able to request larger or smaller units of instructional programs depending upon the generality or specificity of his requests. And that the preparation of instructional materials would be further simplified if the author of the instructional materials could prepare an instructional module and later incorporate that module into a larger module.

The Coursewriter I language was extended to include a submodule call thus permitting any module to be used as a submodule of any other module. It is important that a submodule is not a subroutine of an instructional program but rather an independent instructional program available for execution apart from its use as a submodule. Thus the submodule call introduced here is significantly different from the subroutine call encountered in other languages for writing instructional programs (Zinn, 1968).

The addition of the submodule call, which is an $L^2$ statement, extended level 2 control to the computer. The full capability of embedding tutorial mode processes within other tutorial mode processes is now possible under either student or computer control.

So the revised design called for a single script and a collection of related modules within a given topic area.
Indexing Instructional Modules

The modular approach led to a second problem. It was not unreasonable to provide an instructional program to list available lessons when there were relatively few lessons and once the student selected a lesson he could be expected to remain in that lesson for an entire session. But it was not reasonable to list all of the modules every time the student might wish to select another module. So we were faced with the issue of indexing the modules.

At first the solution to this problem appeared to be printing a module name with every response in the discussion mode. The student could then shift the mockup to tutorial mode and select that module to obtain the desired information.

An experiment was run in which the only information available in discussion mode was module names. A subject was able to locate the modules, but he spent much of his time shifting the system to tutorial mode to obtain a short answer and then returning to discussion mode.

The script was then modified to contain short responses and module names. The student would then need to enter tutorial mode only to obtain additional information. Experimentation with this script demonstrated that subjects did not use tutorial mode. They would remain in discussion mode asking very general questions and receiving very general answers when the suggested module would supply the additional information they needed. This indicated that there were circumstances in which the computer should assume information control in response to a question.
In analyzing the situation, it became clear that there were general questions and specific questions relative to the information available within a given topic area. A specific question was defined as one which could be answered by a single response. It was determined that the computer should respond to such a question and remain in discussion mode. A general question, on the other hand, was defined as a question answered by an instructional module. In response to this type of question it appeared that the computer should automatically shift to tutorial mode and administer the module. Because modules may contain submodules which may themselves contain submodules, the amount of instructional material administered depends upon the generality of the question.

The objective in designing this version of the system was to supply the most specific statement available which is consistent with the student input. This may be a single response in discussion mode, a small module of instructional material or a large module composed of many submodules.

The rationale behind this approach is that a student capable of asking specific questions can effectively control the order of presentation of materials. But a student asking general questions does not know enough about the topic to control the order of presentation, therefore he should surrender information control while the computer answers his question and tests his understanding of that answer. The student may suspend that module to ask additional questions which may lead to the execution of other modules as required. The student may
reject a module selected by the computer; it is not yet clear whether or not this is desirable.

The system was modified and a new script prepared to permit the machine to automatically shift to tutorial mode in response to general questions. But experimentation with this version of the system immediately demonstrated that the automatic shift was not always desirable. It essentially destroyed the discussion mode because of the frequent shift to tutorial mode.

It appears that there are circumstances in which the automatic shift is desirable and others in which it is not. Because it was not at all clear when the automatic shift was useful, a student-controlled switch was introduced to enable and disable this feature. When the mode control switch is on, the machine will instigate a shift to tutorial mode for general questions as described above, and when the switch is off the machine will give a short response to all questions and remain in discussion mode.

The effect of the mode control switch is to determine which mode will predominate. When the automatic shift is enabled, tutorial mode tends to be the dominant mode. The student enters discussion mode either to ask specific questions or to cause another module to be administered. However, when the shift is disabled the student tends to remain in discussion mode.

When the computer automatically shifts to the tutorial mode, it is exerting control at the second level, control. When the student turns the switch on and off he exerts control because he determines
whether or not the computer shall exert control. Thus the introduction of the control switch is the first step toward the extension of GITIT to a system with four levels of control, and the control commands which enable and disable the automatic shift to tutorial mode are statements in the $L^3$ language.

The final version of the GITIT system includes a mode control switch and commands which permit the student to turn the switch on or off as required. Experimentation with this system is discussed in the following chapter.

**Backward and Forward Pointers**

In the conventional approach to programmed instruction, the student starts at the beginning of a lesson which is a comparatively long segment of instructional material and moves through subsequent material under control of the instructional program. The instructor preparing the lesson must anticipate student needs and prepare appropriate paths. The student must follow a preprogrammed path.

The modular approach permits the student to select his own path through the instructional materials. But there is a difficulty in this approach. The student must know what is available to be able to select a path. It would seem that the student can find his way through a topic area only after he knows what the area covers, and that once he knows that he may no longer need the instructional materials. This difficulty was resolved by using backward and forward pointers.

The recognition process in discussion mode is based on key words and phrases. Therefore the student can obtain meaningful responses and
reference to instructional modules only if he uses the vocabulary recognized within the topic area.

In the tutorial mode, the first statement in a module includes a list of prerequisites. This serves as a backward pointer. The last frame of a module suggests topics to be considered next which serves as a forward pointer. The backward and forward pointers are words or phrases recognized in discussion mode. The Coursewriter I language was further extended to accommodate forward and backward pointers.

Backward pointers are not needed in discussion mode, because each response tends to contain other words or phrases recognized within the topic area. So the student may move backward by asking about terms appearing in the response.

It was first assumed that providing forward pointers in the tutorial mode was adequate, but empirical results indicate that this is not the case. Students frequently rejected a module before completion, returned to discussion mode, and could not think what to ask next. So the memory stack was used to provide forward pointers in discussion mode.

Whenever the student asks a general question which causes an instructional module to be selected, a statement suggesting an appropriate next topic may be stored on the memory stack. At a later time, when the computer does not detect any key words or phrases in the student input, this response will be given.

By use of the forward and backward pointers, a student can find his way through the instructional materials. But he must enter a
recognizable key word or phrase to enter the network of forward and backward pointers.

A facility was added to supply the initial word. If the student enters a statement or question not recognized by the computer, and there are no responses on the memory stack, the computer will select a key word or phrase and make a statement using it. By using the suggested word or phrase, the student will obtain a beginning response or module. He may continue to find his way through the available materials by using the backward and forward pointers. Therefore, it is possible to browse and to locate information within a topic area without prior knowledge of the vocabulary of the area.

The Data Structure for GITIT

The mockup used two independent data structures for the two modes. The discussion mode used a script as used in the ELIZA program, and the tutorial mode used a collection of instructional programs written in Coursewriter I. GITIT uses a combined data structure which retains characteristics of both of the original data structures.

The data structure requirements for the two modes are different. In the tutorial mode, the computer must control the order of presentation of material. Therefore, information concerning this order must be available. But it is not necessary to recognize a large variety of student responses because the \( L^0 \) communication events are controlled by the computer and only those responses appropriate to the given statement or question must be recognized.
In the discussion mode the locus of information control is with the student, therefore any question or statement which is appropriate to the topic area must be recognized at any time. But it is not necessary to represent an ordering among the possible responses.

These differences in requirements dictate differences in representation and processing. List processing techniques and complex pattern recognition capability are needed in the discussion mode. But conventional file processing techniques are adequate for the tutorial mode.

The resulting data structure for a given topic area, called a local data structure, includes a collection of instructional modules and a script which is a list structure containing the patterns to be matched against statements and questions entered by the student. The modules are stored as sequentially organized disk files. Each pattern either yields a module element which causes a shift to tutorial mode for execution of the specified module, or a frame reference which causes selected lines from a module to be printed.

The large volume of textual material is thus on disk and transferred to core, one line at a time, as it is required for processing. Since data can be transferred from disk more rapidly than it can be printed on the terminal, the time to transfer lines from disk is not critical. The list structure is comparatively short and can be retained in core.
The Resulting CAI System

GITIT is a CAI system with three levels of control designed to operate within a specified topic area. The language $L^0$ includes a specified technical vocabulary for that topic area.

There are two modes of operation: discussion mode and tutorial mode. Locus of information control is with the student when the system is in discussion mode and with the computer when the system is in tutorial mode. Control is shared by the student and the computer.

The system is initially in discussion mode. If the student asks a specific question or makes a specific statement the computer will respond and remain in discussion mode. A response includes the name of an instructional module containing further information.

The student may exert control at level 2 to place the system in tutorial mode and select an instructional module. The student may also exert control at level 3 to turn on the mode control switch thus enabling the automatic shift to tutorial mode. Then, if the student asks a question or makes a statement which the computer judges to be general, the computer will exert control at the second level causing the system to enter the tutorial mode and an appropriate instructional program will be executed. At the conclusion of an instructional program, the computer will exert control at the second level to return to the previous mode.

A dialogue in discussion mode or the execution of an instructional program in tutorial mode is a level 1 process. Any level 1 process may be terminated by the student, or the student may cause it
to be suspended and a new level 1 process initiated. Upon termination of a process, the process in which it is embedded is resumed.

Submodule calls in instructional programs provide a capability whereby the author of the instructional material may concatenate or embed the level 1 processes resulting from the execution of submodules. The computer exerts control at level 2 when it executes a submodule call.

The student may use the special commands provided to initiate, suspend or terminate a level 1 process thereby concatenating or embedding as he wishes. Thus he may follow paths through the instructional materials which were not anticipated by the author of the instructional programs.

Preparing instructional materials for GITIT is quite simple. The instructor writes the individual modules in a prescribed manner. He also lists the words and phrases which are to be considered a part of the technical vocabulary in the area. An automatic procedure is provided to generate a discussion mode script from the modules and the technical vocabulary. This involves a special automatic indexing process. A detailed prescription for generating instructional materials is given in Chapter VII.

The data structure for a given topic area, called a local data structure, contains a collection of instructional modules and the associated script. To avoid redundantly storing statements, the script contains references to records within instructional modules. A further description of GITIT and its data structure is given in Chapter VI.
The present version of GITIT uses two levels of memory. The script is contained in core. The instructional modules are stored on disk and transferred to core as they are required for execution. The design concept includes extension to three levels of memory to support a version of GITIT having four levels of control, including control of the topic area. Only the local data structure for the present topic area will be on disk, the other local data structures remaining on tape or other forms of mass memory until they are needed. The proposed extension is noted in Chapter VIII.

The GITIT system has several advantages over the previous systems discussed in Chapter III. The system makes efficient use of computer processing time. Experimental runs requiring 30 to 40 minutes of student-computer interaction have used 20 to 30 seconds of processing time. This makes utilization of the system in a time sharing environment feasible, and suggests that this concept might be appropriate for use on a mini-computer.

The efficiency of the program is partially due to the lack of redundancy in the data structure and the effective use of levels of computer memory. Primary computer memory (core) and list processing techniques are used to provide the recognition capability in the discussion mode, but secondary memory (disk) and sequential file techniques are used for machine responses and for instructional modules. Because the script is in core and contains a pointer to a specific disk record, response time is not excessive. By matching the type of memory and processing techniques used to the requirements of man-machine
interaction, economy in the use of the computer system is attained without detrimental effect to system performance.

The computer program is completely independent of the topic area, all information pertaining to a particular topic area being contained in the script and the instructional modules. The system can thus be used for other topic areas without reprogramming.

The modular construction of instructional programs and the automatic generation of the script make the preparation of instructional materials quick and simple. The avoidance of the use of elaborate branching techniques in instructional programs eliminates the need for extensive debugging and pretesting of instructional materials. This approach seems necessary in order to develop a cost-effective system.

The resulting system is efficient with respect to utilization of computer memory, utilization of computer processing time, and the preparation of instructional materials.

*In partial fulfillment of requirements for a course, a graduate student has prepared a set of instructional materials according to the prescription given in Chapter VII. He found the procedure very simple and accomplished the task in a short period of time. The resulting materials, which have not yet been tested, appear quite useable (Coulter, 1972).
CHAPTER V

THE USE OF GITIT TO TEACH A SET
OF DEFINITIONS: A CASE STUDY

GITIT was used to teach a set of definitions to five subjects all of whom were graduate students in the School of Information and Computer Science. The basic experiment extended over two consecutive days and involved student interaction with the system on the second day. This interaction is called Run 1. Supplementary data were gathered for certain students. This additional experimentation and testing required four days with student interaction on one or, at the student's request, two of these days. These interactions are called Run 2 and Run 3.

The instructional materials used for the case study are given in Appendix A; a description of the experiment and a complete record of one student's interaction with the system are given in Appendix B.

In this chapter we present an analysis of the data obtained from the case study with attention to some effects of locus of control in simple and complex tasks and to the adequacy of the GITIT data structure to support the required interaction. After discussing the general finding of this study, we present as a case study a detailed analysis of the performance of one successful and one unsuccessful student.

* The subjects are referred to as students to avoid confusion between the experimental subject and the subject matter.
It is concluded that the present data structure is adequate to support the free learning activities of a student who is able to validly evaluate his own performance, but that additional structure would be required to provide both more feedback to the student and extended machine control for students incapable of valid self-evaluation.

Strategies for Teaching and Learning

In a recent research report, Pask (Pask, 1972) described a series of experiments in which he studied uncertainty regulation in human learning. In addition to prescribed teaching strategies, Pask considered conversational teaching strategies and a free learning situation.

When a prescribed teaching strategy was used, the student proceeded through the subject matter in an order predetermined by the instructor incorporating, at most, simple feedback regulation. In our terminology we would say that locus of information control was with the machine or with the instructor who prepared the instructional materials. This corresponds to the tutorial mode in the GITIT system when it is used independently rather than in conjunction with the discussion mode.

The conversational teaching strategy was designed to regulate the presentation of materials as required by a specific student under the guidance of an instructor. The present version of GITIT cannot accommodate a conversational strategy, however straightforward and relatively simple system modifications would yield this capability. A further discussion of these modifications is presented in Chapter VIII in connection with suggestions for further study.
In Pask's free learning situation the student was free to proceed through material in his own manner within the constraints imposed by certain regulatory conditions. Thus the free learning situation most nearly approximates the situation in our experiment with the exception that our students were not subject to the constraint of any regulatory conditions. Therefore, the experimentation reported here involves a free learning situation somewhat freer than that reported by Pask.

Pask concluded that students taught by a conversational strategy were more successful than those taught by a prescribed strategy, and that students taught by a prescribed strategy were more successful than the free learning students. But there were two important exceptions to this general finding: (1) Some of the students who followed a prescribed strategy which was appropriate for them fared as well as all of the conversational students, and (2) certain of the free learning students were as successful as many of the students taught by a conversational strategy. The success of these exceptional free learning students was attributed to the student having learned how to learn, and it was further claimed that students could be taught how to learn by use of a conversational strategy or an appropriate CAI system. We will show that with minor modification the GITIT system is such a system.

Given Pask's findings, we would expect wide variation in performance in our experimentation due to the amount of freedom afforded the student. This did, in fact, occur. We would further expect that a student who had learned how to learn would be successful in this
situation and we did encounter at least one such subject.*

Simple and Complex Tasks

It is to be expected that the effects of locus of information control would be different for simple and complex tasks, therefore it is important to distinguish between these two types of tasks. We will consider a task in which the student is concerned with only one concept at a time to be a simple task. By a complex task we mean a task in which a student must consider the relationships among a multiplicity of concepts.

The simplest task which we might wish to consider would be the retrieval and rote learning of a set of unrelated items such as vocabulary learning in which the set of terms occurring in the definienda is disjoint of the set of terms occurring in the defiens. A specific example of such a task would be teaching biomedical terminology to nonprofessional personnel where the terms involved are all defined using a non-technical vocabulary. Another example would be teaching French words using English language definitions.

If such a task is to be performed under student control the system needs to recognize the term in the vocabulary and produce the associated definition. This could be accomplished entirely in discussion mode as there is no need for the machine to assume control to explain. Thus the data structure requirements involve an adequate set of patterns to recognize the terms in the vocabulary and a means of identifying the correct definition.

*As will be explained in a subsequent section, one criteria for success was based on supplemental data which were gathered only for selected subjects.
This simple task could also be performed entirely under machine control using tutorial mode to drill the student. Conventional CAI has been extensively used for such drill and test procedures. There is no need for a question-answering capability because the items are unrelated, therefore learning one definition does not give rise to questions about another.

GITIT provides the necessary control and an adequate data structure to support such a simple task. The script provides recognition of words and phrases and a pointer to an appropriate frame which would contain the definition. Adequate instructions are provided for instructional programs to permit drill and test procedures.

A more complex task still involving only one concept at a time would be learning an interrelated set of items such as a technical vocabulary in which the items are defined in terms of each other yielding a chain of definitions. If the student is required to understand each definition rather than learn it by rote, he needs to ask about any unfamiliar terms encountered in a definition. He may also require explanation if the definitions are sufficiently complex that understanding is not immediate upon reading the definition.

There are many examples of such tasks. The terms in any technical vocabulary tend to be defined in such a way that other terms from the technical vocabulary occur in the definition. These definitions also tend to be sufficiently complex that they need to be explained rather than merely stated.

To use discussion mode for such a task, the system must recognize the terms in the vocabulary and supply the appropriate definition.
But it is also necessary to indicate the prerequisite definitions and to provide access to tutorial materials. While using the tutorial materials the student may need to ask about other terms. This may, but need not, follow the embedding structure of the recursive definitions themselves, but "is defined in terms of" is clearly a relation which holds between terms in the vocabulary and might need to be reflected in the data structure.

Thus the data structure needs to include the information required to recognize terms in the vocabulary and match them with appropriate definitions and PI modules. Because the items are interrelated it needs to include back pointers to inform the student of prerequisite materials. Other terminology may arise within the tutorial materials in which case these terms should be recognized by the system to facilitate question-answering.

Use of tutorial mode for such a task requires that information be available to the machine concerning the order of presentation of materials. This includes both the order of frames within an elementary process, i.e. the order of execution of instructions within a instructional program, and a pointer to a next program.

We will demonstrate the adequacy of both the control capability and the data structure of GITIT to support the interaction required for this task for students who know how to learn in a free learning situation. Addition machine control is indicated for students who have not learned how to learn.

A complex task involving multiple concepts might be to require students to investigate relationships between concepts other than one.
concept being used in the definition of another. We shall also show that GITIT is adequate for this task.

Browsing is also a complex task. In the tasks considered above we assumed that the student had a precise set of terms to be learned and could thus ask specifically for the required information. But the student might have only a vague idea what he wanted to know. In this case he would need the machine to suggest topics to him. This requires that the system be able to supply useful suggestions when it does not recognize terms from the vocabulary. Although the experimental task reported here did not require this capability, we will show that GITIT is adequate to support this type of task.

In the experimental task the students were required to learn an interrelated set of definitions. They were given a list of terms from the Computer Science area. After making an initial attempt to define the terms, the students were permitted to utilize the system in any way they wished in order to obtain and learn the definition of these terms. Supplemental data gathered from certain of the students involved questions concerning the similarities and differences between related concepts and a suggested order in which the concepts might be learned.

We shall first consider the experimental task in some detail and discuss the effects of locus of information control on successful performance in this task. Then we shall give consideration to some effects of locus of information control on other tasks.
Criteria for Successful Performance

The task we consider here is learning an interrelated set of verbal definitions. Each student was given a list of 17 terms. The student first attempted to define each term and rated his definition using the usual letter grades A, B, C, D and F. The student then used GITIT to improve his definitions. Each new or modified definition was graded in the same way. The student could also change the grade of an existing definition if he wished to do so. This was accomplished on Day 2 of the experiment and the resulting interaction with the System is referred to as Run 1. (See Appendix B.)

The confidence estimate $\alpha_i(k)$ was determined from the grade assigned by the student for each term $t_i$ on occasion $k$ as follows:

$$
\alpha_i(k) = \begin{cases} 
1.00 & \text{if the grade is A} \\
.75 & \text{if the grade is B} \\
.50 & \text{if the grade is C} \\
.25 & \text{if the grade is D} \\
.00 & \text{if the grade is F}
\end{cases}
$$

The confidence estimate does not, of course, reflect the correctness of the definition. To determine whether or not the definitions were correct, and to establish a numerical measure of this correctness, each definition was graded in the following way.

The correct definition of each term was analyzed to determine the essential characteristics of a correct definition. Then each definition
given by a student was analyzed to determine which characteristics were present. The ratio of the characteristics present in the student's definition to the number of characteristics available in the correct definition was taken to be the measure of correctness of the definition. The correctness of the definition of term $t_i$ on occasion $k$ will be written $\theta_i(k)$ and called the rectitude of the definition.

For purposes of analysis, the terms were divided into two sets: major terms and minor terms. Major terms are the topic of a PI module. Minor terms are not the topic of a PI module but are used within one or more modules. Major terms are considered to be more important because they are essential to the understanding of other terms. Major and minor terms are discussed further in Chapter VII.

The confidence estimate $\alpha_i$ and the rectitude $\theta_i$ for major terms prior to Run 1 are shown in Table 1. The confidence estimate and rectitude at the end of Run 1 are shown in Table 2. Tables 3 and 4 give the corresponding data for minor terms.

**Rectitude of Final Definitions as a Criteria for Success**

A student might be considered successful in the simple task of learning the definitions if he produced correct definitions of the terms by the end of Run 1. Because students were encouraged to write their own definition rather than copying the definition given by the system, and most definitions were given in the form of a paraphrase, it was possible for a student to obtain a correct definition and give an incorrect paraphrase.

*All definitions were analyzed by two graders and the correctness determined by agreement.*
Table 1. Confidence Estimate $\alpha_1$ and Rectitude $\theta_1$ for Major Terms Before Run 1

<table>
<thead>
<tr>
<th>Major Term</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$\theta$</td>
<td>$\alpha$</td>
<td>$\theta$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>field</td>
<td>.00</td>
<td>.00</td>
<td>.25</td>
<td>.00</td>
<td>.50</td>
</tr>
<tr>
<td>node</td>
<td>.00</td>
<td>.00</td>
<td>.50</td>
<td>.50</td>
<td>.50</td>
</tr>
<tr>
<td>linear list</td>
<td>.00</td>
<td>.00</td>
<td>.75</td>
<td>1.00</td>
<td>.25</td>
</tr>
<tr>
<td>link</td>
<td>.50</td>
<td>.25</td>
<td>.50</td>
<td>.25</td>
<td>.25</td>
</tr>
<tr>
<td>access</td>
<td>.75</td>
<td>.50</td>
<td>.75</td>
<td>.25</td>
<td>.25</td>
</tr>
<tr>
<td>allocation</td>
<td>.75</td>
<td>1.00</td>
<td>.50</td>
<td>.25</td>
<td>.25</td>
</tr>
<tr>
<td>stack</td>
<td>.75</td>
<td>.67</td>
<td>.50</td>
<td>.67</td>
<td>.75</td>
</tr>
<tr>
<td>Average</td>
<td>.39</td>
<td>.35</td>
<td>.54</td>
<td>.42</td>
<td>.39</td>
</tr>
</tbody>
</table>

Rank Correlation

|         | .96 | .62 | -.07 | -   | .62 |
Table 2. Confidence Estimate $\alpha_j$ and Rectitude $\theta_j$ for Major Terms After Run 1

<table>
<thead>
<tr>
<th>Major Term</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>field</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>node</td>
<td>1.00</td>
<td>.50</td>
<td>1.00</td>
<td>1.00</td>
<td>.25</td>
</tr>
<tr>
<td>linear list</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>link</td>
<td>1.00</td>
<td>.25</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>access</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>.00</td>
</tr>
<tr>
<td>allocation</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>.50</td>
</tr>
<tr>
<td>stack</td>
<td>1.00</td>
<td>.67</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>Average</td>
<td>1.00</td>
<td>.77</td>
<td>1.00</td>
<td>.86</td>
<td>.52</td>
</tr>
</tbody>
</table>

Rank Correlation

- .16
Table 3. Confidence Estimate $a_i$ and Rectitude $\theta_i$ for Minor Terms Before Run 1

<table>
<thead>
<tr>
<th>Minor Term</th>
<th>$S_1$ $\alpha$ $\theta$</th>
<th>$S_2$ $\alpha$ $\theta$</th>
<th>$S_3$ $\alpha$ $\theta$</th>
<th>$S_4$ $\alpha$ $\theta$</th>
<th>$S_5$ $\alpha$ $\theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>bead</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.00 0.00 0.75 1.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
</tr>
<tr>
<td>pop</td>
<td>0.75 1.00 0.00 0.00</td>
<td>0.00 0.00 0.25 1.00</td>
<td>1.00 1.00 1.00 1.00</td>
<td>0.75 1.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
</tr>
<tr>
<td>LIFO</td>
<td>0.75 1.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>1.00 1.00 1.00 1.00</td>
<td>0.75 1.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
</tr>
<tr>
<td>push down</td>
<td>0.75 1.00 1.00 1.00</td>
<td>0.75 1.00 1.00 1.00</td>
<td>1.00 1.00 1.00 1.00</td>
<td>0.75 1.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
</tr>
<tr>
<td>empty node</td>
<td>0.00 0.00 0.75 0.00</td>
<td>0.00 0.00 0.25 0.00</td>
<td>0.00 0.00 1.00 1.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.25 0.00 0.00 0.00</td>
</tr>
<tr>
<td>FIFO</td>
<td>0.75 1.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>1.00 1.00 1.00 1.00</td>
<td>0.75 1.00 0.00 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
</tr>
<tr>
<td>record</td>
<td>0.50 0.00 0.50 0.50</td>
<td>0.50 0.00 0.50 0.50</td>
<td>0.50 0.00 1.00 1.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.75 1.00 0.00 0.00</td>
</tr>
<tr>
<td>address of a field</td>
<td>0.00 0.00 0.50 0.00</td>
<td>0.00 0.00 0.00 0.00</td>
<td>0.00 0.00 0.75 1.00</td>
<td>0.00 0.00 0.50 0.00</td>
<td>0.50 0.00 0.00 0.00</td>
</tr>
<tr>
<td>dummy field</td>
<td>0.00 0.00 0.25 0.00</td>
<td>0.00 0.00 1.00 1.00</td>
<td>1.00 1.00 1.00 1.00</td>
<td>0.00 0.00 0.50 0.00</td>
<td>0.00 0.00 1.00 1.00</td>
</tr>
<tr>
<td>ends of a list</td>
<td>0.00 0.00 0.25 0.50</td>
<td>0.00 0.00 0.00 1.00</td>
<td>1.00 1.00 1.00 1.00</td>
<td>0.00 0.00 0.50 0.00</td>
<td>0.00 0.00 1.00 1.00</td>
</tr>
<tr>
<td>Average</td>
<td>0.35 0.40 0.33 0.20</td>
<td>0.23 0.40 0.95 0.70</td>
<td>0.50 0.63</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Rank Correlation: 0.93 0.78 0.55 0.21 0.57
Table 4. Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Minor Terms After Run 1

<table>
<thead>
<tr>
<th>Minor Terms</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$\theta$</td>
<td>$\alpha$</td>
<td>$\theta$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>bead</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>pop</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
</tr>
<tr>
<td>LIFO</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
</tr>
<tr>
<td>push down</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.50</td>
</tr>
<tr>
<td>empty node</td>
<td>1.00</td>
<td>.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
</tr>
<tr>
<td>FIFO</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>record</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
</tr>
<tr>
<td>address of a field</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
</tr>
<tr>
<td>dummy field</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>ends of a list</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>.50</td>
<td>1.00</td>
</tr>
<tr>
<td>Average</td>
<td>1.00</td>
<td>.90</td>
<td>.95</td>
<td>.95</td>
<td>.83</td>
</tr>
</tbody>
</table>
Considering the average rectitude on major terms given in Table 2 we find that \( S_2 \) and \( S_5 \) had a rectitude of 1.00 thus were 100 per cent successful. \( S_1, S_3 \) and \( S_4 \) were less successful with average rectitude of .77, .67 and .61, respectively. Taking 70 per cent as the criteria for success, we would conclude that \( S_2 \) and \( S_5 \) were very successful, \( S_1 \) moderately successful and \( S_3 \) and \( S_4 \) unsuccessful. All students were successful on minor terms since the average rectitude was at least .90 for all given students.

Supplemental data which were gathered for three of the students required an additional four days of experimentation as explained in Appendix B. The students were given another list of terms to define and again used GITIT to obtain the definitions. The confidence estimate and rectitude for the definitions given for major terms before and after Run 2 are shown in Tables 5 and 6. All three students have a final average rectitude greater than .70 thus we would consider that all three were successful. The corresponding data for minor terms given in Tables 7 and 8 indicate that \( S_1 \) and \( S_3 \) were successful on minor terms but \( S_2 \) was not.

The students were then asked to state the similarities and differences between certain paired concepts. As can be seen from the results given in Table 9, both \( S_1 \) and \( S_2 \) were able to give valid comparisons; \( S_3 \) was not.

Several of the definitions of major terms were modified as a result of this task. Table 10 gives the confidence estimate and rectitude of the final definitions given for all major terms reflecting these
Table 5. Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Major Terms Before Run 2

<table>
<thead>
<tr>
<th>Major Term</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$\theta$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>link field</td>
<td>.50</td>
<td>1.00</td>
<td>.00</td>
</tr>
<tr>
<td>insert</td>
<td>.75</td>
<td>1.00</td>
<td>.50</td>
</tr>
<tr>
<td>delete</td>
<td>.75</td>
<td>.75</td>
<td>.50</td>
</tr>
<tr>
<td>sequential allocation</td>
<td>.75</td>
<td>.75</td>
<td>.25</td>
</tr>
<tr>
<td>linked allocation</td>
<td>.50</td>
<td>.75</td>
<td>.50</td>
</tr>
<tr>
<td>queue</td>
<td>1.00</td>
<td>.50</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>.71</td>
<td>.79</td>
<td>.46</td>
</tr>
<tr>
<td><strong>Rank Correlation</strong></td>
<td>-.36</td>
<td>.77</td>
<td>.49</td>
</tr>
</tbody>
</table>

Table 6. Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Major Terms After Run 2

<table>
<thead>
<tr>
<th>Major Term</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha$</td>
<td>$\theta$</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>link field</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>insert</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>delete</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>sequential allocation</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>linked allocation</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>queue</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>1.00</td>
<td>.92</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Rank Correlation</strong></td>
<td>-</td>
<td>-</td>
<td>.54</td>
</tr>
</tbody>
</table>
### Table 7. Confidence Estimate $a_i$ and Rectitude $\theta_i$ for Minor Terms Before Run 2

<table>
<thead>
<tr>
<th>Minor Term</th>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_1^\alpha\theta$</td>
</tr>
<tr>
<td>linked list</td>
<td>.75</td>
</tr>
<tr>
<td>name of a field</td>
<td>.00</td>
</tr>
<tr>
<td>empty list</td>
<td>.50</td>
</tr>
<tr>
<td>contents of a node</td>
<td>.25</td>
</tr>
<tr>
<td>name of a node</td>
<td>.00</td>
</tr>
<tr>
<td>Average</td>
<td>.30</td>
</tr>
<tr>
<td>Rank Correlation</td>
<td>.72</td>
</tr>
</tbody>
</table>

### Table 8. Confidence Estimate $a_i$ and Rectitude $\theta_i$ for Minor Terms After Run 2

<table>
<thead>
<tr>
<th>Minor Terms</th>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_1^\alpha\theta$</td>
</tr>
<tr>
<td>linked list</td>
<td>1.00</td>
</tr>
<tr>
<td>name of a field</td>
<td>1.00</td>
</tr>
<tr>
<td>empty list</td>
<td>1.00</td>
</tr>
<tr>
<td>contents of a node</td>
<td>1.00</td>
</tr>
<tr>
<td>name of a node</td>
<td>1.00</td>
</tr>
<tr>
<td>Average</td>
<td>1.00</td>
</tr>
<tr>
<td>Rank Correlation</td>
<td>-</td>
</tr>
</tbody>
</table>
Table 9. Confidence Estimate $\alpha_i$ and Rectitude $\theta_i$ for Comparison of Terms

<table>
<thead>
<tr>
<th>Terms Compared</th>
<th>$S_1 \alpha$</th>
<th>$S_1 \theta$</th>
<th>$S_2 \alpha$</th>
<th>$S_2 \theta$</th>
<th>$S_3 \alpha$</th>
<th>$S_3 \theta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>node vs. field</td>
<td>1.00</td>
<td>.80</td>
<td>.75</td>
<td>.60</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>linked vs. sequential allocation</td>
<td>.75</td>
<td>.60</td>
<td>1.00</td>
<td>.80</td>
<td>.75</td>
<td>.60</td>
</tr>
<tr>
<td>accessing vs. inserting</td>
<td>1.00</td>
<td>.90</td>
<td>.75</td>
<td>.90</td>
<td>1.00</td>
<td>.00</td>
</tr>
<tr>
<td>accessing vs. deleting</td>
<td>1.00</td>
<td>.90</td>
<td>.75</td>
<td>.90</td>
<td>.75</td>
<td>.00</td>
</tr>
<tr>
<td>stack vs. queue</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>.67</td>
</tr>
<tr>
<td>name of a field vs. content of a field</td>
<td>1.00</td>
<td>1.00</td>
<td>.50</td>
<td>.90</td>
<td>.50</td>
<td>.33</td>
</tr>
<tr>
<td>name of a field vs. name of a node</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
<td>.75</td>
<td>.80</td>
</tr>
<tr>
<td>linked list vs. linear list</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
<td>.90</td>
</tr>
<tr>
<td>Average</td>
<td>.97</td>
<td>.90</td>
<td>.81</td>
<td>.89</td>
<td>.78</td>
<td>.54</td>
</tr>
</tbody>
</table>
Table 10. Confidence Estimate $a_i$ and Rectitude $\theta_i$ for Final Definitions of All Major Terms

<table>
<thead>
<tr>
<th>Term</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a$</td>
<td>$\theta$</td>
<td>$a$</td>
</tr>
<tr>
<td>field</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>node</td>
<td>1.00</td>
<td>.50</td>
<td>1.00</td>
</tr>
<tr>
<td>linear list</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>link</td>
<td>.75</td>
<td>.25</td>
<td>1.00</td>
</tr>
<tr>
<td>link field</td>
<td>1.00</td>
<td>1.00</td>
<td>.75</td>
</tr>
<tr>
<td>access</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>insert</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>delete</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>allocation</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>sequential allocation</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>linked allocation</td>
<td>1.00</td>
<td>.75</td>
<td>1.00</td>
</tr>
<tr>
<td>stack</td>
<td>1.00</td>
<td>.67</td>
<td>1.00</td>
</tr>
<tr>
<td>queue</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>.98</td>
<td>.84</td>
<td>.98</td>
</tr>
<tr>
<td><strong>Correlation</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
modifications. Based on the rectitude of their final definitions, $S_2$ was most successful with a rectitude of .92, $S_1$ slightly less successful with a rectitude of .84 and $S_3$ clearly unsuccessful with a rectitude of .62.

The confidence estimate and rectitude of the final definitions of minor terms are as shown in Tables 4 and 5. The overall average for minor terms is as follows:

$$
S_1: \ a = 1.00 \quad \theta = .93
$$

$$
S_2: \ a = .90 \quad \theta = .83
$$

$$
S_3: \ a = .80 \quad \theta = .92
$$

Thus all three students were, on the average, successful with the minor terms.

According to Pask's criteria for learning, the rectitude of the written definitions is not an adequate measure of learning. In his recent research report (Pask, 1972) Pask states his view as follows:

Concept reconstruction is believed to lie at the root of the memory process (at least for this type of subject matter); in fact, stating the matter strongly, a memory of $R_i$ is the reconstruction of a concept for $R_i$. Hence the fact that a student can give a correct explanation of $R_i$ when he is asked to do so is taken to indicate that he has a concept of $R_i$ and is able to reconstruct it, i.e. that he has a memory of $R_i$.

Pask used a mechanical device to test a student's knowledge of Probability theory. The "explanation" of a concept involved setting up the device in an appropriate way to solve a problem. We decided that
an analogous task would be to ask students to generate examples. This requires a student to reconstruct a concept in a specific context.

**Ability to Generate Examples as a Criteria for Successful Learning**

After approximately one week the three students from whom supplemental data were obtained were recalled and asked to give examples of four of the major terms. The rationale was that if the students had merely learned a verbal definition by rote they would be unable to generate an example. On the other hand, the ability to produce an example indicates an understanding of the underlying concept. The examples were graded for validity and the rectitude determined. These values are given in Table 11.

**Table 11. Rectitude of Examples**

<table>
<thead>
<tr>
<th>Major Term</th>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_1$</td>
</tr>
<tr>
<td>linear list</td>
<td>1.00</td>
</tr>
<tr>
<td>linked allocation</td>
<td>1.00</td>
</tr>
<tr>
<td>stack</td>
<td>1.00</td>
</tr>
<tr>
<td>queue</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>1.00</td>
</tr>
</tbody>
</table>

We found that $S_1$ was able to give valid specific examples of all four concepts. Both $S_2$ and $S_3$ had considerable difficulty thinking of examples, and when they did give an answer it tended to be inappropriate.
The definitions of all terms rest ultimately on the concept of a node which is defined in terms of computer words, thus the only appropriate context is internal machine representation of data. Neither $S_2$ nor $S_3$ had had sufficient experience in computer programming to be aware of possible examples in this context. $S_2$ several times repeated the definition itself as an example and was unable to be very specific. $S_2$ and $S_3$ attempted to give examples from automata theory thus becoming more, rather than less, abstract and failing to produce a valid example. We attributed this difficulty to limited experience and coursework in the Computer Science area.

**Recall of Definitions as a Criteria of Success**

To confirm that the students had learned the verbal definitions, they were again recalled. This time they were asked to write definitions of three major and two minor terms without referring to their notes. The rectitude for these terms is given in Table 12.

<table>
<thead>
<tr>
<th>Term</th>
<th>STUDENT</th>
<th>STUDENT</th>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
</tr>
<tr>
<td>node</td>
<td>1.00</td>
<td>1.00</td>
<td>.50</td>
</tr>
<tr>
<td>linear list</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>queue</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>linked list</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>dummy field</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>1.00</strong></td>
<td><strong>1.00</strong></td>
<td><strong>.90</strong></td>
</tr>
</tbody>
</table>
It is interesting to note that all subjects did well on recall. $S_3$ was the only subject who made an error in a definition, and this still indicated improvement over the definition given by $S_3$ at the end of Run 1.

We concluded that students can learn verbal definitions using the GITIT system, but that students lacking appropriate background are unable to apply the verbal definitions to generate examples. Future versions of GITIT should include an entrance test to assure that students are aware of the context of the material they are to learn and that they have the necessary prerequisite knowledge to deal with the material. An example checking facility is also needed to assure that the students understand the concepts and are able to apply them to an appropriate area.

The Importance of Valid Self-Evaluation in Free Learning

In the free learning situation reported here, the student has control over the teaching/learning procedure. Thus he must be able to evaluate his own performance in order to effectively exert control.

A student may already know a definition and be convinced that he is right; in this instance no further learning of the definition is required. Or the student may have a correct concept in mind but be uncertain of its correctness in which case the learning is simple as he needs only to confirm the correctness of present concept. If a student does not know a definition and recognizes that he does not we would anticipate that he would be able to learn it. It is the student who
has an incorrect concept but is mistakenly convinced that he is right who might be expected to encounter difficulty in a free learning situation because he may never investigate the topic to discover that he is wrong.

Therefore we might expect that a correlation between the student's subjective grading and the correctness of his definitions would be necessary for effective learning. To determine if this was the case, the correlation between the confidence estimate and the rectitude was calculated for the definitions given for major terms prior to Run 1. The correlation coefficient is shown in Table 1. The correlation for S₁ is .96 which is significant at the 1 per cent level. The correlation is .62 for both S₂ and S₅ which, although it is not significant even at the 10 per cent level, does suggest some agreement between the student's subjective grading and the correctness of the definitions given. However, the correlation for S₃ is -.07 indicating no correlation between his performance and his self-evaluation. The correlation for S₄ was not calculated because there was no variance in his confidence estimate, but the fact that his subjective grading was unrealistically high indicates that he was not capable of valid self-evaluation.

Considering only the rectitude at the end of Run 1, we found that S₁, the student with the highest correlation, was moderately successful. S₂ and S₅ were highly successful and had low but positive

---

*The Spearman coefficient of rank correlation was used (Ferguson, 1966).
correlation, and S₃ and S₄ were unsuccessful and showed no correlation. The supplemental data gathered for S₁, S₂ and S₃ indicated that S₁ was highly successful at generating examples and both S₂ and S₃ totally unsuccessful.

The correlation coefficient was also calculated for S₃ for the definitions given for major terms at the end of Run 1. (The lack of variance in confidence estimate made the calculation meaningless for the other students.) As can be seen in Table 2, the correlation coefficient is .16 indicating that S₃ was still unable to evaluate his performance at the end of the run.

The correlation coefficients for minor terms before Run 1 were also calculated and were shown in Table 3. The correlation for S₁ is again significant at the 1 per cent level; the correlation for S₂ is significant at the 2 per cent level and the correlation for S₅ is significant at the 10 per cent level. The correlation for S₃ is not significant but is very nearly so; the correlation for S₄ is quite low. All students had high average rectitude on the minor terms at the end of Run 1. None of the students were required to give examples of minor terms thus we have no other criteria for determining successful performance on minor terms.

In order to consider the relationship between the student's confidence and his correctness during the run, we computed the average confidence estimate α and average rectitude θ for major terms at each stop. These computations were made for each of the five students and the results are shown graphically in Figures 7-11 for S₁-S₅, respectively.
Figure 7. Average Confidence α Estimate and Rectitude θ on Major Terms for S₁ During Run 1
Figure 8. Average Confidence Estimate $\alpha$ and Rectitude $\theta$ on Major Terms for $S_2$ During Run 1
Figure 9. Average Confidence Estimate $\alpha$ and Rectitude $\theta$ on Major Terms for $S_3$ During Run 1
Figure 10. Average Confidence Estimate $\alpha$ and Rectitude $\theta$ on Major Terms for $S_4$ During Run 1
Figure 11. Average Confidence Estimate $\alpha$ and Rectitude $\theta$ on Major Terms for $S_5$ During Run 1
Because only the major terms were considered, the curves are flat whenever the student was working with minor terms. These graphs therefore reflect the difference in strategy as well as the validity of the student's judgment; strategy differences will be considered in the next section.

S\textsuperscript{1} shows a good initial agreement between confidence and rectitude (see Figure 7), but his confidence increases more rapidly than does his rectitude especially at the end of the run. He had become convinced that his mistaken concepts of several terms were correct. There was no feedback during the run to bring this error to his attention. He did, however, discover some of his error later and correct his definitions during a subsequent run.

S\textsuperscript{2} and S\textsuperscript{5} both showed a high degree of correlation between confidence and rectitude throughout the run (see Figures 8 and 11). In both instances the initial confidence estimate was slightly higher than the rectitude, but this discrepancy was corrected before the end of the run.

S\textsuperscript{3} and S\textsuperscript{4} both showed wide discrepancies between confidence and rectitude. S\textsuperscript{3} had a good initial agreement on average confidence estimate and rectitude for the major terms but an extremely low correlation between confidence estimate and rectitude for individual terms (see Figure 9). He was aware that he did not know the material, but he was unaware of where his weaknesses were. His confidence estimate increased during the run but this was not accompanied by an increase in rectitude resulting in a wide discrepancy between the final rectitude
and confidence estimates. $S_4$ had an unrealistically high initial confidence estimate, and it remained high throughout the run. (See Figure 10.) Neither $S_3$ nor $S_4$ show much increase in rectitude during the run.

The correlation between confidence estimate and rectitude was also calculated for the major terms prior to Run 2 for those students for whom additional data were gathered. These values were shown in Table 5.

In this instance $S_1$ had an extremely low correlation (-.36) despite the fact that he was quite successful. Note, however, that his average rectitude at the beginning of the run was quite high and, because of the low variance, the discrepancy between his confidence estimate and his rectitude was not great. It is also significant that, with one exception, his confidence estimate was less than or equal to his rectitude.

$S_2$ and $S_3$ have somewhat higher correlation although in neither instance is the correlation significant. Both were moderately successful based on average rectitude at the end of the run. Neither was later able to generate examples although they were able to recall the definition of the terms.

At the end of Run 2, the three students were asked to state the similarities and differences between several paired concepts including stack and queue. Both $S_1$ and $S_3$ had mistaken concepts of stacks. Both students considered that a stack was more general concept than a queue and that queues were special types of stacks.
This confusion is thought to arise from the existence of an unnamed concept. Stacks and queues are both linear lists in which processing occurs only at the end of the list. For stacks all processing occurs at one end of the list while for queues insertions are made at one end of the list and deletions are made at the other. Clearly stacks and queues taken together form a class of linear lists which might be called input-output restricted lists but was not given a name in the instructional materials. Both S₁ and S₃ considered "stack" to be the name of this unnamed concept, and had the correct concept of a queue.

When they were asked to state the similarities and differences between stacks and queues, S₁ detected an inconsistency in his definitions and elected to use the system again to clarify his concepts of stack and queue. He correctly compared the two. S₃ did not detect the error and remained convinced a queue was a special type of a stack. Feedback concerning the correctness or incorrectness of the comparison might have caused S₃ to recognize his error.

Thus we conclude that valid self-evaluation is important if a student is to succeed in a free learning situation. Not all students are able to validly evaluate their own performance in the absence of feedback. Some students became convinced that they were correct when in fact they were mistaken. Other students had a correct verbal definition but were unable to generate an example. Both of these problems could be solved by the incorporation of an example checking device into the GITIT system. Students could then be required to demonstrate their
comprehension of a concept by giving an example. This would provide feedback to the student permitting him to better evaluate his comprehension of the subject matter and would provide information whereby the system could exert control over student behavior.

The need for an example checking device is in accord with Pask's findings. In the research discussed previously (Pask, 1972), all students except the free learning students were required to give a non-verbal explanation of a topic before leaving the topic. Pask concluded that this was essential to assure that students did understand the concept. Thus our findings are in agreement with Pask's conclusions as to the necessity for a student to explain or otherwise demonstrate his understanding of a concept.

**The Regulation of Uncertainty**

Pask also concluded that uncertainty regulation is a prerequisite for effective learning. To test Pask's findings in this regard, we computed uncertainty based upon the student's subjective grading of his definitions.

A student's uncertainty \( H_i(k) \) concerning his definition of term \( t_i \) on occasion \( k \) is computed from the grade currently assigned to the definition by the student as follows:

\[
H_i(k) = \begin{cases} 
.00 & \text{if the grade is A} \\
.25 & \text{if the grade is B} \\
.50 & \text{if the grade is C} \\
.75 & \text{if the grade is D} \\
1.00 & \text{if the grade is F}
\end{cases}
\]
Note that \( H_i(k) = 1.00 - \alpha_i(k) \).

The uncertainty of a set of terms is taken to be the sum of the uncertainties of the individual terms. Thus the uncertainty of a single term ranges from 0 to 1.0 and the uncertainty of a set of \( n \) terms ranges from 0 to \( n \). This uncertainty measure for a set of terms was chosen to reflect the increase in task uncertainty for students who are working on a number of terms simultaneously rather than processing the terms one at a time.

It is clear from analysis of the protocols of Run 1 that students did not work on the entire set of 17 terms at once, but rather that they selected one or a few terms to work on, perhaps enlarging the set to include related terms as they proceeded. We call the set of terms on which the student is working his working on set. A term enters the working on set at the time at which the student asks for information about the term or receives and attends to unrequested information concerning the term. It was assumed that the student attended to unrequested information if he used it to modify his definitions.

The precise criteria for determining the membership of the working on set are based upon the concept of student processes as introduced in Chapter II. A student process is assumed to be oriented toward a learning goal and may have embedded in it subprocesses oriented toward subgoals. In this instance the learning goals involve the definitions of the various terms. The criteria are presented here without further explanation. Student processes are considered in further detail in conjunction with the case study later in this chapter, and the reader
is asked to defer detailed consideration of these criteria until that
time.

The membership of the working on set is determined as follows:

1. All terms whose definition or grades are modified by the
student at stop \( k \) are in the working on set at stop \( k \).

2. All terms modified at stop \( k \) which were discussed prior to
stop \( j < k \) are considered to be in the working on set at stop \( j \) and at
all intervening stops except as provided by item 4 below.

3. If any term in the working on set at stop \( k \) serves as the
goal of a process within which are embedded subprocesses, then any term
which serves as the goal of one of these subprocesses is also in the
working on set at stop \( k \) provided the subprocess was not terminated
prior to stop \( k - 1 \).

4. Once a term enters the working on set it remains in the set
until either (a) the final mention or modification of that term or (b)
the termination of the process associated with that term.

The student was permitted to stop the system at any time to modify his
definitions and grades, and it was only at these stops that modifica­
tions occurred. Therefore we take these stops to be the occasions on
which we compute uncertainty. We use occasion \( k = 0 \) to denote the time
immediately preceding the run.

For the task of learning the definitions of a set of \( n \) terms we
introduce the following terminology and notation.

The total task uncertainty on occasion \( k \geq 0 \) is defined as
The reduction in uncertainty on occasion \( k \geq 0 \) is defined as

\[
\Delta H(k) = H^*(k-1) - H^*(k)
\]

The working on set on occasion \( k > 0 \) is denoted \( W_k \). The residual uncertainty on occasion \( k \) is defined as

\[
\hat{H}(k) = \sum_{t_i \in W_k} H_i(k)
\]

where

\[
W = \bigcup_{j=1}^{k} W_j
\]

is the set of all terms which either are or have been in the working on set.

The total task uncertainty at each stop during Run 1 for each of the five students is shown graphically in Figure 12. We see that the uncertainty for \( S_3 \) was initially quite high and it dropped but did not reach zero. The uncertainty for \( S_1 \) was not as high at the beginning of the run, dropped about the same amount and did reach zero. \( S_2 \) and \( S_5 \) each had a moderately low initial uncertainty; \( S_2 \) had a final uncertainty of zero and \( S_5 \) a final uncertainty near zero. \( S_5 \) had little uncertainty at the beginning of the run and failed to reduce the uncertainty.
Figure 12. Total Task Uncertainty $H_0^*$ for All Five Students on Run 1
Pask found that the failure to control uncertainty resulted from incomplete concepts of certain topics, that is, topics which the student could not adequately explain. The difficulty arose when the unexplained topic was used to build a new concept.

To determine whether or not a student did abandon a topic before he thought that he adequately understood the topic, the reduction in uncertainty $\Delta H(k)$ and the residual uncertainty $H^*(k)$ at each stop in Run 1 was computed for each of the five students. $\Delta H(k)$ and $H^*(k)$ are shown in Figures 13-17. The size of the working on set and the number of communication events prior to each stop are also shown in these figures.

We would expect that uncertainty associated with major terms would cause far greater difficulty than uncertainty associated with minor terms. Therefore, the uncertainty associated with major terms is shaded in the graphs of $\Delta H$ and $H^*$ in Figures 13a-17a and 13b-17b. Major terms are also indicated by shading in the graph of the size of the working on set in Figures 13c-17c. In the graph of the number of communication events (Figures 13d-17d) the shading indicates discussion mode.

It is apparent from these graphs that the students used the system in entirely different ways. $S_1$, who was successful, started with a fairly large working on set and, as a result, a high $\Delta H(1)$. He concentrated first on major terms, reducing most of the associated uncertainty to zero. He then worked on minor terms until, at stop 12, $\Delta H(12) = 0$. He then worked on the remaining terms but did not build up
a. Reduction in Uncertainty ($\Delta H$)

b. Residual Uncertainty ($\tilde{H}$)

c. Size of Working on Set

d. Number of Communication Events

Figure 13. Graphical Data for $S_1$
Figure 14. Graphical Data for $S_2$
Figure 15. Graphical Data for $S_3$
Figure 16. Graphical Data for $S_4$
Figure 17. Graphical Data for $S_5$
any residual uncertainty during the end of the run. He also used tutorial mode far more than any other student.

S₂ used an entirely different overall strategy. He worked on one term at a time not leaving any residual uncertainty until the last few frames. He used the discussion mode predominately. S₂ was successful based on final rectitude and recall but was unable to generate examples.

S₃, who was unsuccessful, did not control his uncertainty. His residual uncertainty was high throughout most of the run. It is perhaps significant that he began with minor terms without first learning the underlying major terms. He tended to have a small working on set and to use the discussion mode predominately.

S₄ had little uncertainty at the outset and failed to reduce it. In fact, ΔH(9), ΔH(12) and ΔH(13) are negative indicating an increase in uncertainty for minor terms.

S₅ failed to control uncertainty on minor terms, but left very little residual uncertainty on major terms. Like S₁, she worked on major terms, then (related) minor terms. But unlike S₁, she had a small working on set most of the time and used discussion mode predominately.

It is clear that the students used the system in radically different ways. Some students were able to control their uncertainty while others were not. The student who was clearly successful, S₁, drove ΔH to zero and the student who was clearly unsuccessful, S₃, had a very high ΔH throughout the run. This is in agreement with Pask's findings.

Clearly some students left topics before they drove their uncertainty to zero. Thus they would attempt to use incomplete concepts to
build other concepts. But uncontrolled uncertainty can result from another source not indicated in these graphs.

The students were free to proceed through the materials in any order they wished. They were not required even to consider prerequisite concepts. Thus, for example, if a student tried to learn the definition of a stack without asking about the terms occurring in the definition of "stack," the uncertainty associated with these terms is not reflected in AH since they have never been in the working on set. Thus we need to consider the order in which the various topics were studied.

Student Strategies

All five students were presented with the same experimental task, but the students displayed different learning strategies in that they proceeded through the materials in entirely different ways. We will briefly consider the strategies of all five subjects on Run 1, and then analyze in detail the strategies employed by one successful student (S1) and one unsuccessful student (S3).

The items to be learned were highly interrelated. A dependence matrix for the major terms learned during Run 1 is shown in Figure 18. (This matrix was derived from the dependence matrix for the entire set of experimental materials which is presented in Chapter VII.)

There are three types of relationships between terms shown in this matrix. Term A is said to have a type I dependency on term B if term B occurs in the definition of term A. For example, "field" has a type I dependency on "node" because a field is defined as a named portion of a node.
Figure 18. Dependence Matrix for Major Terms Learned During Run 1

Term A is said to have a type II dependency on term B if term B occurs in the PI module for term A but not in its definition. A type III dependency arises from the use of a term in a submodule. S/M is used to indicate that a term is the major term of a submodule. See Chapter VII for a more complete discussion of dependence matrices.

Type I, II, and III dependencies are represented in the data structure by back pointers. Because the first frame in each PI module contains the definition of the major term, a type I dependency is represented by a back pointer from the first frame. Type II and III dependencies are represented by back pointers from subsequent frames.

The important thing to note in the dependence matrix is that every term has either a type I or type II dependency on "node," and "field" is the major term for a submodule of the module on nodes.
Thus an understanding of nodes and fields is essential to understanding the other terms.

A precedence graph for the major terms is shown in Figure 19. Solid lines represent a primary precedence relation which results from a type I dependency and dotted lines represent a secondary precedence relation resulting from a type II and III dependency. The secondary precedence relations are not significant when the student is in discussion mode because it is only in the explanation of the term in the tutorial module that the precedent term occurs.

Not all of the students attempted to learn the major terms in an order which respects these precedence relations. Table 13 shows the order in which definitions of major terms were first changed.

Table 13. Order of Processing of Major Terms During Run 1

<table>
<thead>
<tr>
<th></th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S_3$</th>
<th>$S_4$</th>
<th>$S_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1.5</td>
<td>2.5</td>
</tr>
<tr>
<td>Field</td>
<td>2</td>
<td>5</td>
<td>5.5</td>
<td>6</td>
<td>2.5</td>
</tr>
<tr>
<td>Link</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Linear List</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>1.5</td>
<td>5</td>
</tr>
<tr>
<td>Allocation</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Access</td>
<td>5</td>
<td>4</td>
<td>5.5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Stack</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Figure 19. Precedence Graph for Major Terms Learned During Run 1
The terms are listed in the order in which an "ideal" subject might process the terms, that is, an order which preserves the precedence relations.

We see that $S_1$ started with "node" proceeded to "field" then "linear list" and "stack." The only primary precedence violated here was the dependence of stack upon access, and $S_1$ had a high (.75) initial confidence estimate on access.

The ordering in Table 12 does not reflect the order in which students first asked about terms. $S_1$ first asked about stacks, then linear lists and then nodes following backpointers from each term to the precedent term. He then worked with node and field and related minor terms before returning to linear list and finally to stack. Next he asked about "access" and followed a back pointer to "allocation" after recording his definition of access. He proceeded from "allocation" to "linked allocation" (not required on this run) and then to "link."

Obviously $S_1$ did use the back pointers and he did treat the task as that of learning interrelated items. And he did so successfully. He is clearly a student who has learned how to learn. And GITIT provided him with adequate information so that he could learn the concepts in an order approximating that indicated by primary precedence relations despite the fact that he was not shown the precedence graph or the depending matrix.

Another subject, $S_2$, appears to have treated the task as a rote learning task. He took the terms in precisely the same order in which
they were presented to him, and asked a question containing each term. In most cases he immediately corrected his definition as required and proceeded to the next term.

S₃ attempted to learn interrelations among items but he did not display a successful strategy for doing so. He asked first about minor terms occurring in the module on stacks then considered stacks. He failed to follow the backpointers from "stack" to precedent terms, thus the first major item he dealt with was the last major term in the precedence graph.

He then changed his definitions of "node" and "linear list" based upon information presented about stacks. Never during the entire first run did he see the definitions of "node" or "field," and he saw the definition of "linear list" after processing all other items.

S₃ next asked about "allocation" and modified his definitions of "access" and "field" based upon the use of these terms in the material on allocation. Then he asked about links and finally about linear lists. It appears that S₃ had not learned how to learn in this type of situation.

The next subject, S₄, might be expected to display a different approach to using the system because he was using GITIT for review. He had learned the definitions almost a year earlier in an undergraduate course, and he had been using the concepts extensively in subsequent coursework. As a result he was confident that he knew the definitions as reflected by his initial subjective grades. (See Tables 1 and 3.)

This student found himself in the position of knowing more about the subject than the teacher, i.e. GITIT. Through familiarity with the
subject matter the student had apparently developed his own cognitive structure of the area which did not entirely agree with definitions in the system. He frequently obtained a definition from the system only to decide that his definition was better. He essentially had a different set of equally correct concepts for the same area, but in determining the rectitude of his definitions they were judged incorrect because they were not the definitions taught by the system.

His initial behavior represented a looking around at a more general level. He asked about "data structures" and "list structures" before considering "linear list" and following the backpointer to "node." He then followed a relationship existing in his own cognitive structure of the area and asked how one would access a node. He next followed a backpointer to "allocation" then followed a backpointer from "allocation" to "consecutive words of computer memory" which was a primitive. He then asked about "stacks" and then about "accessing stacks" and followed a backpointer to "operations." He continued to follow backpointers and to ask about terms which he considered to be related, many of which were not on the list of terms to be learned and some of which were not in the system.

Thus S treated the task as a complex task of learning a network of interrelated concepts. He was not successful. (Rectitude = 60 percent). It is postulated that his failure is attributable to the fact that he had more information on the subject area in his cognitive structure than was contained in the GITIT data structure, thus causing him to reject the definitions presented to him in favor of his own.
A careful inspection of the definitions he gave revealed that they were entirely reasonable and quite interconsistent and in some instances closer to common usage than the definitions in the system.

The last subject, $S_5$, was also quite familiar with the subject area from previous coursework, but she had not previously been exposed to this particular set of definitions.

Her initial behavior represented an attempt to determine the particular context with which we were concerned. She asked, "What kind of allocation can be made?" and modified her definition of allocation to make it more specific. She then asked, "What kind of access is referred to?" and followed a backpointer from "access" to "node." She worked on "node" and "field" returned to "access" then considered "linear list" "link" and "stack" in that order.

$S_5$ tended to move from one term to a related term sometimes following backpointers and at other times asking next about terms she knew to be related although these relationships were not necessarily represented in the GITIT data structure. On two occasions she asked how one concept was related to another. Thus we conclude that $S_5$ approached the task as a complex task of learning relationships between concepts rather than a rote learning task.

The students chose different paths through the instructional materials, not all of which were equally effective for the students. However, it is clear that the GITIT system is sufficiently flexible to permit students to use different strategies.

The students also differed widely in their use of the control capability of the system. Figure 20 shows the embedding structure of
Figure 20. Embedding Structure for all Students on Run 1
the machine processes for each of the five students. Whenever one machine process is embedded in another, the embedded process is shown below the process in which it is embedded. Shading is used to indicate discussion mode processes. The length of the line representing each process is proportional to the number of communication events which occurred in the process.

S₁ was on the system far longer than any other student and he made extensive use of tutorial mode. S₁ also used the embedding capability more than did other students.

S₂ and S₃ both had very short runs with shallow embedding and minimal use of tutorial mode. It is interesting that the only PI modules which S₂ used were those on "node" and "field" and S₃ did not use this material at all. Since node and field are concepts which are basic to the understanding of other concepts, this could explain the fact that S₂ was more successful than S₁.

Both S₄ and S₅ used the embedding capability although S₅ did so only at the outset. Both of these students made limited use of the tutorial mode.

Table 14 gives a summary of the number and percentage of communication events in discussion and tutorial mode for each of the five students. It is clear from this table and from the preceding graph that the five free learning students utilized the system quite differently.
Table 14. Number of Communication Events in Discussion and Tutorial Mode During Run 1

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>S_1</th>
<th>S_2</th>
<th>S_3</th>
<th>S_4</th>
<th>S_5</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>21</td>
<td>17</td>
<td>13</td>
<td>36</td>
<td>31</td>
<td>118</td>
</tr>
<tr>
<td>%</td>
<td>29</td>
<td>61</td>
<td>43</td>
<td>75</td>
<td>74</td>
<td>53</td>
</tr>
<tr>
<td>n</td>
<td>52</td>
<td>11</td>
<td>17</td>
<td>57</td>
<td>11</td>
<td>103</td>
</tr>
<tr>
<td>%</td>
<td>71</td>
<td>39</td>
<td>57</td>
<td>25</td>
<td>26</td>
<td>47</td>
</tr>
<tr>
<td>TOTAL</td>
<td>73</td>
<td>30</td>
<td>48</td>
<td>42</td>
<td>221</td>
<td></td>
</tr>
</tbody>
</table>

A Case Study of One Successful and One Unsuccessful Student

In this section we will analyze the strategies employed by one successful student, S_1, and one unsuccessful student, S_3. Both students had as their goal learning a set of definitions. But the two students decomposed this goal into different subgoals producing different processes within the student and within the computer.

A process was defined in Chapter II as a program in execution. Both the computer and the student are assumed to be capable of multiprogramming and to be executing programs at 0th, first, second, and third levels of control. These processes are denoted P^0, P^1, P^2 and P^3 where the superscript denotes the level of control. Multiple processes at the same level are differentiated by subscripts.

The processes with which we are concerned here are those at the levels 0 and 1. The machine processes at these levels are indicated on the protocol for Runs 1, 2 and 3 for student S_1 given in Appendix B.
The machine processes are not given for student S₃.

Following the convention established in Chapter II, we will show student processes by means of a goal decomposition diagram. In order to demonstrate the method of analysis, we will consider the processes of S₁ during Run 1 in some detail, and then discuss selected items from other runs.

Student Processes for S₁

The following analysis of student processes is based on the protocol of the interaction of student S₁ during Run 1 (given in Appendix B), on the definitions which the student gave and modified as the experiment progressed (also given in Appendix B) and on interviews with the student.

The student's primary goal was determined by the experimental situation in which he has given a list of terms to define. However, he was given complete freedom to pursue these definitions in any order he wished, thus he determined his own subgoals. In analyzing the student's processes we shall attempt to identify the resulting subgoals and their associated subprocesses. The resulting goal-directed student processes and subprocesses occur at the first level of control, therefore we will consider only the level 1 processes.

The student processes at the first level of control for student S₁ during Run 1 are shown in the goal decomposition diagram in Figure 21. In this diagram the corresponding machine process or processes for each goal or subgoal are given in parentheses.

The initial level 1 machine process P₁ was a non-controlling process, and the corresponding student process is a controlling process...
Figure 21. Goal Decomposition Diagram for $S_1$ During Run 1
Figure 21. (Continued)
directed toward the goal of learning the 17 terms. The student's first subgoal, \( G_1 \), was to obtain the definition of a stack. Thus, the second level 1 process is a controlling process directed toward this goal. A stack is defined in terms of a linear list, and the student was not sure of the definition of a linear list. Therefore, he immediately embedded the process directed toward the goal \( G_{1,1} \) of learning the definition of a linear list. This in turn led to a need to know the definition of a node, hence goal \( G_{1,1,1} \). These three level 1 student processes correspond to the level 0 machine processes \( P^0_1, P^0_2 \) and \( P^0_3 \) within \( P^1_1 \), thus the student's embedding structure for these processes is not reflected within the embedding structure of the machine processes.

It is clear that these student processes are controlling processes and that the student was using a selection strategy. The student next initiated the controlling machine process \( P^1_2 \), the execution of the PI module NODE. If the strategies are to be compatible the student should employ a reception strategy yielding a non-controlling student process. However, the data indicate that the student continued to use a selection strategy extracting from the information presented by the machine only that which he needed. Definitions of record and bead were obviously ignored as the student requested this information later. This interpretation was confirmed by the subject in a later interview in which he said that he was only attending to information necessary to the understanding of the definition of stack. Therefore, we interpret the data such that the student is using a selection strategy during the first three frames of the module NODE and that his controlling process
continues to be active during this time.

The fourth and fifth frames of the module NODE have to do with empty nodes. The student evidently did attend to this information because he recorded the definition of an empty node at the next stop. Thus a non-controlling student process corresponding to the controlling machine process resulting from the concatenation of $P^0_7$ and $P^0_8$ was directed toward the goal of learning the definition of an empty node.

The student then initiated a controlling process to obtain the definition of a field. This process arose because both a node and an empty node were defined in terms of fields. After obtaining the definition of a field in discussion mode, he caused the system to enter tutorial mode to administer the PI module. The data do not indicate whether he used a selection or a reception strategy during this time, therefore we will assume that he used a reception strategy. At this time the student stopped to record his definitions. This is Stop 1 of Run 1.

The embedding is now six levels deep as can be seen in Figure 21. This appears to exceed the limitations of immediate memory in the student because he was unable to record all of the information which he had obtained.

At Stop 1 the student correctly recorded the definition of a node and incorrectly recorded the definition of an empty node, changing the rating on both definitions from F to A. (It is not known which definition he recorded first.) He defined an empty node as "a node that contains zeroes or garbage" despite the fact that he had correctly answered
"no" to the question "Can a node be empty?" This indicates that a student can answer questions correctly and convince himself that he understands when in fact he does not. The student did not discover this error during the course of the experiment.

Next, the student attempted to record the definition of a field. He was unable to do so because, as he commented to the experimenter at the time, he had lost the information. According to the model of embedding of processes we would say that this is because he had terminated the process concerned with fields to return to the suspended processes on nodes and empty nodes. It is interesting to speculate as to whether or not he could have recorded all three definitions had he first recorded the definition of a field.

The student then reinitiated the PI module FIELD. It appears that the student used a selection strategy which was not compatible with the concurrent controlling machine process $P_4^1$. In the first frame the student received the information he needed concerning fields and stopped to record this definition. At Stop 2 the student recorded a partially correct definition of a field which he rated C. Then, based upon his own processing and without further interaction with the system, he recorded a correct definition containing more information than was presented within the current execution of the PI module. This suggests that when he lost the information he did not forget it entirely, and the limited information in the first frame of FIELD served to remind him.

At this point the student asked for a repetition of the definition of a stack. In a later interview he explained that he had
forgotten why he needed to know the definition of a node in order to understand the definition of a stack. So he restarted the embedding process.

This is the same phenomena which was observed in the preliminary experimentation with the mockup and is assumed to be the result of severe human limitations on immediate memory. The student was unable to utilize the prompting capability incorporated in GITIT to compensate for this limitation because his embedding structure was not reflected by the embedding of machine processes.

Had the mode control switch been turned on at the beginning of the run, the student's questions concerning stacks, linear lists, nodes and fields would each have caused an automatic shift to tutorial mode. Controlling machine processes corresponding to the student processes would have been suspended enabling the machine to remind the student of suspended processes. This suggests that it might be desirable to have the mode control switch on when the student first enters a new topic area. The possibility of utilizing the mode control switch in this manner should be investigated in future research.

The student asked about stacks and linear lists and then elected to use the module on linear lists. The data give no indication to the contrary, therefore we assume that the student now employed a reception strategy during his process oriented toward $G_{1,1}$.

The first frame of the module LLIST specifies that linear order is a prerequisite concept to that frame. Thus the student suspended his process and the machine process $P_{5}^{1}$ to enter the discussion mode and ask
about linear order, returning to the suspended processes after one $L^0$ communication event. The processes corresponding to the execution of LLIST then continued.

During the execution of LLIST a system failure occurred on the machine and the student recorded his definition of a linear list at that time. When the system came up the experimenter drove the system to the same state thus we assume that the student and machine processes continued. After two frames the system again failed and the student decided that he did not need the remainder of the module LLIST.

After processing resumed, the student again asked for the definition of a stack. It is not clear whether he needed the definition again because of the interruption due to the system failures or because he had exceeded his immediate memory capacity. The student now understood what a linear list was and did not ask about linear lists. Instead he asked about the end of the list, another term occurring in the definition of a stack. After receiving his response which was a frame in the module STACK he elected to see that module. We assume that the student used a reception strategy.

After a few frames the student realized that he did not understand where insertions and deletions occurred for a stack. So he placed the machine in discussion mode ($P^1_{g}$). He stated his misconception and received a response which confused him further because it had to do with linear lists in general rather than stacks in particular. Because he did not notice this distinction it appeared to confirm his mistaken concept of a stack.
He stopped (Stop 4) to reread his material in an effort to understand the difficulty. He recognized an inconsistency in the information he had, but could not decide what the problem was. Part of his problem was an initial false concept of a stack. He did not completely correct this problem until Day 5.

At this point he terminated machine processes $P_7^1$ and $P_8^1$ to restart the module on stacks. We interpret this as a continuation of his process oriented toward $G_1$. After the first frame of the module STACK, the student stopped (Stop 5) to modify his definition of a stack, but the definition he gave is incorrect and inconsistent with the information in the current frame. He rated his definition B, the same as the rating of his initial definition. He also modified his definition of the ends of a linear list. He continued through the remainder of the module until the machine termination of the module but did not alter his definition. He stopped to record the definitions of lifo and pop as they occurred, Stops 6 and 7. This is taken as indication that he was using a reception strategy for these terms.

The student then asked a question about the relationship of pop and push down to LIFO stacks. Since all stacks are LIFO stacks, this question and the subsequent one about LIFO stacks reflect the student's confusion about the definition of a stack which was mentioned above. He recorded the definition of push down at Stop 8.

The student next asked what a bead was and recorded the definition (Stop 9).

Then the student placed the system in tutorial mode and selected the module FIELD. At the conclusion of the module he asked about dummy
field and address of a field, and recorded each definition. He later explained that he had scanned the FIELD module for information concerning these two terms.

Next the student asked about FIFO and changed his subjective grade of his definition of a stack to an A. The definition was not correct.

The student then asked about access which lead to allocation, next to linked allocation and then to link. In each instance he followed a back pointer. Finally he asked about record and recorded that definition.

The strategy employed by S₁ during Run 1 reflects extensive use of back pointers. He began with stack, and roughly the first half of the run was used to learn the precedent terms. He employed a reception strategy only when the information presented was closely related to the items he was working on rejecting other information presented in tutorial mode. As noted previously, the only violation of a primary precedence relation during this run involved access which was used in the definition of stack.

The behavior of S₁ during Run 2 was not essentially different. He again made extensive use of back pointers. The goal decomposition diagram for this run is given in Figure 22.

S₁ was the only student who elected to use the machine for a third run. When he attempted to answer the comparative questions concerning stacks and queues he discovered that there was an error in his definition.
Figure 22. Goal Decomposition Diagram for $S_1$ During Run 2
He thought that a queue was a special type of stack, in particular a FIFO stack. His goal during Run 3 was to determine the difference between a stack and a queue and to resolve his uncertainty concerning related minor terms.

In the goal decomposition diagram for Run 3 given in Figure 23 we see that S₁ first asked the machine to define pop again. His definition of pop referred to stacks without qualifications rather than LIFO stack. This was correct but inconsistent with other definitions which referred erroneously to LIFO stacks and FIFO stacks. All stacks are LIFO or last-in-first-out linear lists. Queues are FIFO or first-in-first-out linear lists. "FIFO stack" is a contradiction.
After confirming that pop was defined for stacks and not just LIFO stacks, S₁ requested the module STACK. After two frames he stopped (Stop 1) to modify his definition of stack. His definition was still in error.

After terminating the STACK module, S₁ asked about insertions and deletions for FIFO stacks, a contradiction in terminology. He had asked essentially the same question during Run 1 and been misled by the response because he failed to note that the machine response concerned linear lists in general rather than stacks in particular. He received a different response this time, but it was again concerned with linear lists in general.

Next he asked for a definition of FIFO and was told that a FIFO
list (not stack) was a queue. He next requested the module QUEUE. After one frame he stopped (Stop 2) and correctly modified his definition of queue. After the run he corrected his definitions of LIFO, FIFO, and push down. He did not correct his definition of stack, but he claimed later that this was an oversight and it appears that this is the case since he correctly answered the question concerning the similarities and differences between stacks and queues.

Student Processes for $S_3$

Student $S_3$ also had as his overall goal learning all 17 terms. But his subgoal decomposition was entirely different as can be seen from Figure 24. He began with the minor terms LIFO and FIFO. After receiving the definition of LIFO he guessed the definition of FIFO.

Next he asked if a push down stack was a LIFO. All stacks are push down stacks. $S_3$ had the same misconception of stacks as did $S_1$, and $S_3$ never corrected this error.

After asking again about push down stacks, $S_3$ recorded the definition of pop which is defined in terms of stack despite the fact that he did not know what a stack was.

He then requested the module on stacks and modified his definition of stack after two frames. He did not follow the back pointers from stack, and modified his definitions of stack and linear list based on the way in which these precedent terms were used in the STACK module. He did not see the definition of node during Run 1.

$S_3$ then asked about allocation and again ignored back pointers to linear list and node, modifying his definition of node based on its
Figure 24. Goal Decomposition Diagram for $S_3$ During Run 1
Figure 24. (Continued)
usage in the PI module on allocation. Processing throughout Run 1 proceeded in much the same way.

During Run 2 we turned the control switch on in an attempt to encourage S₃ to utilize tutorial mode. This caused the machine to automatically shift to tutorial mode and begin administering a PI module whenever a major term was recognized. The goal decomposition diagram for this run is given in Figure 25.

S₃ began with "deletion" which was not recognized because this variation of "delete" had not been entered in the script. The machine responded by suggesting several terms which S₃ rejected. When the machine suggested "storage allocation" S₃ responded by asking about linked storage allocation. The machine automatically selected a PI module which S₃ rejected after two frames.

S₃ then asked about queues and immediately rejected the PI module. He next asked about "delete" and rejected the module after three frames. Back pointers were ignored.

The student next asked about the name of a field and did not consider the answer satisfactory, so he selected the NODE module from which the response had been relisted. He had not used this module previously. He also went through the FIELD module which is a submodule of NODE. He did not obtain a definition of the name of a field.

S₃ continued throughout Run 2 ignoring back pointers and for the most part processing items in a seemingly unrelated manner. He did not make a third run.
G: All Terms

G_1: Deletion

G_2: Linked Allocation

G_3: Queue

G_4: Delete

G_5: Name of a Field

G_5,1: Node

G_5,1,1: Field

G_6: Empty List

G_7: Link Field

G_8: Insert

G_9: Name of a Node

G_9,1: Content of a Node

Figure 25. Goal Decomposition Diagram for S_3 During Run 2
Summary

$S_1$ and $S_3$ employed different strategies. $S_1$ followed back pointers and nearly always learned all precedent terms before leaving a term. $S_3$ processed related minor terms together and went from minor terms to the relevant major term. But he ignored back pointers completely.

$S_1$ was much more successful than $S_3$ and apparently had learned how to learn where $S_3$ had not. This suggests that additional machine control is indicated to require students who have not learned how to learn to follow back pointers whenever they do not understand the precedent terms.

The Adequacy of the GITIT Data Structure to Support Simple and Complex Tasks

The GITIT data structure was found to be adequate for both simple and complex tasks when a student used an effective strategy for the task. Additional structure is indicated for students who do not display effective strategies in order to utilize GITIT to teach students to learn.

The Adequacy of Machine Responses

All $L^0$ communication events in the discussion mode were analyzed by two experimenters to determine whether or not the responses were adequate. A distinction was made between responses which were based upon recognition of terms from the vocabulary and those which were not.

Responses which resulted from the recognition of terms from the vocabulary were further divided into two groups: direct responses and
indirect responses. A response was considered to be direct if it was an exact answer to the question. Because the present materials are designed to teach definitions, most questions in this category were of the form "What is a ____" and the response "A ____ is . . .". However, there were other types of questions which also received direct responses including several questions of the form "How is A related to B?" Fifty-four per cent of all machine responses were direct responses to the student's question or statement.

Other responses which were based upon the recognition of terms in the vocabulary were said to be indirect. Indirect responses were rated very good, good, adequate or inadequate. The percentage of responses in each category is shown in Table 15.

An indirect response which was rated very good was usually quite as acceptable as a direct response. Examples of very good indirect responses are the communication events which comprise the machine processes $P_{41}^0$ and $P_{73}^0$ shown in the protocol for Run 1 for $S_1$ given in Appendix B. Examples of good indirect responses may be found in $P_{39}^0$ and $P_{40}^0$ and adequate responses in $P_{24}^0$ and $P_{27}^0$.

A machine response was considered to be at least adequate if the student was given useful information. Eighty per cent of all machine responses were either direct responses or indirect responses which were judged to be adequate or better.

Of the remaining 20 per cent, 11 per cent were inadequate indirect responses. The remaining 9 per cent were in response to a student input which did not contain any term from the vocabulary.
Of this 9 per cent, 4 per cent were considered to be appropriate and 5 per cent were judged inappropriate.

Table 15. The Adequacy of Machine Responses in Discussion Mode

<table>
<thead>
<tr>
<th>Adequacy of Machine Response</th>
<th>STUDENT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$S_1$</td>
</tr>
<tr>
<td></td>
<td>n</td>
</tr>
<tr>
<td>Direct</td>
<td>24</td>
</tr>
<tr>
<td>Indirect Very Good</td>
<td>3</td>
</tr>
<tr>
<td>Indirect Good</td>
<td>2</td>
</tr>
<tr>
<td>Indirect Adequate</td>
<td>3</td>
</tr>
<tr>
<td>Indirect Inadequate</td>
<td>3</td>
</tr>
<tr>
<td>No term recognized Appropriate</td>
<td>1</td>
</tr>
<tr>
<td>No term recognized Inappropriate</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>37</td>
</tr>
</tbody>
</table>

A total of 16 per cent of all responses were unsatisfactory, either inadequate indirect responses or inappropriate responses given when no term was recognized. An analysis of these 26 interactions revealed that 9 per cent were requests for information which was not contained in the instructional materials, 5 per cent resulted from the
omission of one minor term from the vocabulary, the failure to include variations of certain terms (i.e. "deletion" for "delete") and an error in entering the script. The remaining 1 per cent were caused by complex student requests involving multiple terms.

One of these problems was discussed in the previous section. It arose when S asked a question about stacks and received a response about linear lists. It could be resolved by entering "insertions and deletions for stacks" as a minor term. The other two unsatisfactory responses resulted from questions S asked about stack access. This problem could also be eliminated by the addition of a minor term.

In summary, 84 per cent of the responses were satisfactory, and minor modifications to the vocabulary and script would raise this to 90 per cent. The addition of two or three frames to the PI modules would produce additional improvement. We conclude that the GITIT data structure is adequate to produce satisfactory machine responses in discussion mode.

The Adequacy of Back Pointers

The first frame of every PI module contains the definition of a major term and back pointers to all precedent terms used in the definition. If other terms are used within the module, a back pointer to the term is contained in the frame in which it is first used.

It was demonstrated in the previous section that S did effectively follow back pointers and thus learned the material in an order which respected most of the primary precedent relations. The other students did not utilize the back pointers to this extent. Thus, it
appears that the back pointers provided in the GITIT data structure are adequate for the student who is following an effective strategy.

The Adequacy of Forward Pointers

A PI module may contain as its last frame a suggestion of a next topic based on the precedent relationships. Such a frame is called a forward pointer. Whenever a major term is recognized in discussion mode, the forward pointer (if any) from the associated module is placed on the memory stack. This frame is later generated if no terms are recognized in a student input.

The task reported here did not require the use of forward pointers because the student was given the complete list of terms to be learned. But, let us imagine a situation in which the student is simply told to learn as much as he can using the system.

Our hypothetical student might proceed as follows:

WHAT SHALL WE TALK ABOUT?
D: WHAT DO YOU SUGGEST

LETS DISCUSS NODE
(Machine prompting because no term was recognized.)

D: WHAT IS A NODE
A NODE IS . . . [NODE]

D: TUTOR NODE


(Nodes are basic data elements used to construct linear lists and other data structures.)

D: DEFINE LINEAR LIST
A LINEAR LIST IS . . . [LLIST]

D:*TUTOR LLIST

(The LLIST module contains a back pointer to "address of a node.")

T:*DISCUSS

WHAT SHALL WE TALK ABOUT?

D: THE ADDRESS OF A NODE

THE ADDRESS OF A NODE, ALSO CALLED A LINK OR POINTER TO THAT NODE . . . [LINK]

D: HOW ARE THEY USED

THE FACT THAT A POINTER TO ONE NODE CAN BE STORED IN ANOTHER NODE PERMITS US TO REPRESENT COMPLEX DATA STRUCTURES WITHIN A COMPUTER BY THE USE OF WHAT IS CALLED LINKED ALLOCATION. THIS IS ONE TYPE OF STORAGE ALLOCATION WHICH IS USED FOR LINEAR LISTS. 

(A forward pointer from the memory stack.)

D:*RETURN (to LLIST module)

(Continuation of LLIST)

T:*RETURN (to discussion mode)

D: WHAT SHOULD I CONSIDER NEXT

IT IS IMPORTANT TO CONSIDER THE OPERATIONS PERFORMED ON A LINEAR LIST. 

(A forward pointer from the memory stack.)

Our hypothetical student, who knew no terms at the outset, has received pointers to "node," "linear list," "address of a node," "storage allocation" and "operations." If we were to continue to follow the protocol we would find that he receives a forward pointer from "operations" to "stacks and queues." Every module in the structure is
either the module for one of these six major terms or a submodule thereof. Thus, a student can find all materials even though he does not know the vocabulary.

**Adequacy for Simple and Complex Tasks**

GITIT has an adequate pattern recognition capability to recognize a specified vocabulary (including specified variations of terms) and provide a definition. Each definition is printed together with a reference to the appropriate PI module. This is adequate for the simple task of learning unrelated items.

Each definition contains back pointers to precedent terms used in the definition. Whenever additional terms are used within the associated PI module, back pointers to these terms are provided. Since the precedent terms are themselves recognized, this structure is adequate for the more complex task of learning interrelated items provided that the student follows the back pointers.

The system design permits the inclusion of the information required for tasks involving multiple concepts although only a limited amount of this type of information is included in the experimental materials. For example, there is a module on "stacks and queues" which includes the individual modules on "stack" and "queue" as submodules and contains a comparison of the two concepts. The major term for this module is "stacks and queues." Additional relational materials can be incorporated into the structure in this manner. Similarly, the concatenation of two or more (major or minor) terms can be introduced as a minor term. Then the response to any student input containing all of
these terms would be a frame in which they were all mentioned and the relationship discussed.

The data structure is also adequate for the complex task of browsing because forward and backward pointers are provided both in discussion mode and in tutorial mode. The use of the memory stack to provide a forward pointer as well as the prompting (i.e. LETS DISCUSS NODE) capability permits a user who is not familiar with the specified vocabulary to obtain terms which serve as entry points to the available materials.

The data structure in the current version of GITIT is adequate for a wide range of tasks for a user who has an adequate strategy for the task. Strategies are discussed in the next section.

Some Effects of Locus of Control in Simple and Complex Tasks

A student who has learned how to learn can effectively exert control over his learning process, and locus of information control should be with such a student. For a student who has not learned how to learn, locus of information control should be with the machine. When locus of information control is with the machine, the system can be used to teach a student to learn.

Different tasks require different strategies. A student is effective on a given task when he uses an effective strategy. A student who has learned how to learn has a repertoire of learning strategies and the ability to select the strategy appropriate to each task.
Effective Strategies for Learning

The simple rote learning task, i.e. learning a set of unrelated items, requires a simple strategy. The more complex task of learning interrelated items requires a more complex strategy which includes the rote learning strategy as a component. The task of learning the relationship between two items requires a still more complex strategy including this strategy as a component, and so on. Miller et al. (Miller, 1960) introduced the idea of a Test-Operate-Test-Exit or TOTE unit to describe such hierarchical behavior. The basic TOTE unit is shown in Figure 26.

![Figure 26. The Basic TOTE Unit](image)

A TOTE unit for learning the definition of a term is shown in Figure 27. The test element involves a check to see whether or not the definition is known, i.e. the student's evaluation of his uncertainty
concerning the item. The operational unit of learning the definition varies with the complexity of the task.

In a very simple task the operation of learning a definition simply involves obtaining and reading the definition and committing it to memory. If the definition is not self-explanatory, the use of a PI module may be required.

Figure 28 shows a strategy for learning unrelated items. This is precisely the strategy followed by $S_2$ who took each term in the order in which it was presented to him, using the PI module for that term where necessary.

Learning an interrelated set of items requires a much more complex strategy. When a definition is not understood, it is necessary to test the back pointers to precedent terms.
If there are one or more precedent concepts which are not understood, these concepts must be learned before using the tutorial material for the current term. Additional precedent terms may arise in the use of the tutorial materials requiring a continued check for the understanding of precedent terms.
A strategy for learning interrelated items is shown in Figure 29. Here the operational unit LEARN PRECEDENT TERMS is itself a TOTE unit requiring that the definition be tested and learned. It is these embedded TOTE units which produce the embedding of student processes described earlier.

In reviewing the behavior of $S_1$, we see that he used this strategy. This is especially clear in his initial behavior in which he moves from stack to linear list to node. Most of the time $S_1$ was successful. But, when he became convinced that he was correct when in fact he was not, he was not successful. The difficulty is that student control is based on uncertainty—and $S_1$ was successful in driving his task uncertainty to zero—but our criteria for success is based on rectitude which is unknown to the student in the absence of feedback. An example checking device is needed to provide such feedback.

$S_3$ did not follow back pointers. The system could be modified to force a student to follow back pointers by requiring that he demonstrate his understanding of precedent concepts via the example checking device before he is allowed to proceed. This would force the student to follow the prescribed strategy, thus teaching him an effective strategy for learning interrelated items.

Complex tasks can also be described as TOTE units. Learning the relationship between two items involves learning the individual items and then contrasting them. A strategy for this task is shown in Figure 30 where the "LEARN A" and "LEARN B" operational units are each TOTE units as in Figure 29. This was the strategy used by $S_1$ on Run 3 and
Figure 29. A Strategy for Learning Related Items
Figure 30. A Strategy for Learning the Relationship Between Two Items
Using GITIT to Teach Students How to Learn

GITIT can be used to teach strategies to students thus teaching the students how to learn. An example checking facility would be provided and used to provide control information for the machine and feedback to the student.

It is important to note that student control is based on uncertainty and machine control is based on rectitude, and that these two measures are not always correlated. The student's goal is to reduce his uncertainty and the machine's goal is to increase the student's rectitude. The machine can indirectly exert control over the student by increasing the student's uncertainty whenever the student is falsely confident.

A student would be permitted to start at any point. But if he fails to investigate back pointers the machine would assume control and require that he learn all precedent concepts unless he had already demonstrated his mastery of the concepts with the example checking device.

The above procedure is the conversational teaching strategy used by Pask, and he has demonstrated that it can be used to teach students how to learn. GITIT can be readily modified to perform in this manner, thus GITIT can be used to teach students how to learn.

Students would first be taught strategies for simple tasks and machine control relaxed as the student gained mastery of the strategy. The student would then move to more complex tasks, requiring more complex strategies. Locus of control would be with the student for the
component substrategies which he had already mastered in learning the simpler tasks; locus of control would be with the machine otherwise shifting to the student as he learned the strategy.

In terms of the conceptual model presented in Chapter II, instructional processes are programs in execution. Dialogue concerning the subject matter results from the execution of level 0 programs, and learning the content of the subject material involves the construction and modification of level 0 programs. Level 1 programs correspond to strategies. A student who learns how to learn, i.e. develops effective strategies for learning, does so by constructing and storing in his memory level 1 programs corresponding to learning strategies.

The discussion of goals and available strategies and the selection of an appropriate strategy for a given task occurs at level 2. The system modification required to utilize GITIT to teach students to learn would permit the machine to exert control at level 2 until the student developed both an adequate repertoire of strategies for various tasks (level 1 programs) and the facility to select an appropriate strategy for the task at hand (a level 2 program). A student who has learned how to learn is then able to effectively control at the second level.

Conclusion

Locus of information control should be with the student when he has an appropriate strategy for the task at hand; it should be with the machine otherwise. Locus of control should be with the student when he has an adequate repertoire of strategies and is competent to select an appropriate learning goal and strategy, and with the machine if he
does not. Locus of control$^3$ should be with the self-directed learner because it is he who selects the topic area.
CHAPTER VI

A DETAILED DESCRIPTION OF GITIT

An experimental dual-mode CAI system has been designed and implemented on the Burroughs B-5500 computer with user interaction via a teletype terminal. The system, called the Georgia Institute of Technology Instructional Translator (GITIT), includes a conversational routine and a routine which administers a segment of programmed instructional material. When the conversational routine is active, we say that the system is in discussion mode; when the routine which administers programmed instructional material is active, we say that the system is in tutorial mode. Special commands are provided whereby the user may cause the system to shift from one mode to the other.

In discussion mode, the order of presentation of material is controlled by the student. He makes a statement or asks a question and the machine responds with either a statement or a question. The machine then waits for the user to take the initiative and enter another statement or question. Because the presentation of material is determined by the student, we say that the locus of information control is with the student.

In the tutorial mode, the order of presentation of material is based on an instructional program prepared by an instructor and administered by the machine. The student is presented with a few lines of material (usually a definition) and then asked a question concerning the
material. His response is evaluated by comparison with correct and incorrect answers contained in the instructional program. If his answer is judged to be correct he is then presented with new material; if it is not correct he may be given a hint and asked to try again or remedial material may be presented. Because the order of presentation of material is determined by the machine under the control of the instructional program we say that the locus of information control is with the machine.

The important aspect of the system is the facility to shift modes. The mode in which the system is operating may be controlled either by the student or by the machine. When the student is controlling the mode of operation we say that locus of control, \(^2\) i.e. locus of control of locus of information control, is with the student, and when the machine is controlling the mode of operation we say that locus of control \(^2\) is with the machine. Because the student can control whether or not locus of control \(^2\) is with the machine, we say that locus of control \(^3\) is always with the student.

Initially locus of control \(^2\) is with the student and the system is in discussion mode, thus locus of information control is also with the student. If the student wants the system to assume locus of information control and tutor him on some concept, he can cause the machine to enter the tutorial mode and administer a small segment of programmed-instructional material called a programmed-instruction (PI) module. If the system is in tutorial mode he can shift the machine to discussion mode to discuss the material in the PI module, later returning to
tutorial mode to continue with the programmed section. He can also
cause the machine to enter tutorial mode from tutorial mode. In this
case one module is suspended while another module is administered. The
student may later return to the module which was suspended and continue.
There is no fixed limit to the number of modules which may be suspended
at one time. When the machine enters the tutorial mode, a PI module is
selected by the student and the machine begins to administer the
selected module. When the machine returns to the tutorial mode it
resumes the module which was suspended, continuing from the point of
interruption.

There are also conditions under which the machine will instigate
a shift of mode. Whenever the machine completes a PI module it will
automatically exit the tutorial mode and return to the previous mode.
If the system is in discussion mode and the student asks a question
which cannot be answered in a few lines, and locus of control is with
the machine, the machine will shift to tutorial mode and administer an
appropriate PI module. Commands are provided which permit the student
to enable or disable this automatic shift to tutorial mode.

Before considering the mechanism for shifting modes we will
describe the discussion and tutorial modes and their associated data
structures in more detail.

The Tutorial Mode

The routine which administers a programmed-instruction (PI)
module does so by interpretively executing an instructional program
written in an extended version of Coursewriter I, a language initially
developed by IBM (Meadow, 1970). There may be arbitrarily many instructional programs or modules available to the machine. Each module has a unique name. When the machine enters the tutorial mode, the student is asked if he knows the module name. If he does not, a machine controlled dialogue questions him about areas of interest and suggests a module or modules which might be of interest to him. He may select one of the suggested modules. When the student enters a module name the machine will execute the specified instructional program.

The data structure for the tutorial mode is thus a collection of instructional programs, each with a unique name. A formal definition of the syntax of an instructional program and an example of an instructional program are given in Appendix C.

An instructional program consists of a sequence of frames, where a frame contains either a definition, a statement, a question, a submodule call or a forward pointer. When the machine executes an instructional program, it begins with the first frame, and proceeds sequentially through the frames unless it encounters a branch. If a branch is encountered, and the conditions under which the branch is to be taken are met, the machine will next execute the frame referenced by the branch and will continue sequentially from that frame.

A frame consists of a sequence of commands. The present version of the system recognizes the 13 commands shown in Table 16. Any frame may be labeled, in which case the first statement in the frame is a label of the form

\[ \text{LA} \quad \text{label word} \]
Table 16. The Command Language Used for Instructional Programs

<table>
<thead>
<tr>
<th>Operation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA label word</td>
<td>Used to label a frame</td>
</tr>
<tr>
<td>(Label)</td>
<td></td>
</tr>
<tr>
<td>BR label word</td>
<td>Specifies a branch to a corresponding label</td>
</tr>
<tr>
<td>(Branch)</td>
<td></td>
</tr>
<tr>
<td>DF text</td>
<td>The text is typed and machine waits for student to</td>
</tr>
<tr>
<td>(Definition)</td>
<td>respond before continuing</td>
</tr>
<tr>
<td>RD text</td>
<td>Same as DF command</td>
</tr>
<tr>
<td>(Statement)</td>
<td></td>
</tr>
<tr>
<td>CL module name</td>
<td>The module specified is executed and the program</td>
</tr>
<tr>
<td>(Call)</td>
<td>continues</td>
</tr>
<tr>
<td>QU text</td>
<td>The text is typed and machine waits for student to</td>
</tr>
<tr>
<td>(Question)</td>
<td>respond. The response is used to determine the</td>
</tr>
<tr>
<td></td>
<td>next action</td>
</tr>
<tr>
<td>CA text</td>
<td>The text is compared to the student's response</td>
</tr>
<tr>
<td>(Correct answer)</td>
<td></td>
</tr>
<tr>
<td>CB text</td>
<td>The text is compared to the student's response</td>
</tr>
<tr>
<td>(Alternate</td>
<td></td>
</tr>
<tr>
<td>correct answer)</td>
<td></td>
</tr>
<tr>
<td>WA text</td>
<td>The text is compared to the student's response</td>
</tr>
<tr>
<td>(Wrong answer)</td>
<td></td>
</tr>
<tr>
<td>WB text</td>
<td>The text is compared to the student's response</td>
</tr>
<tr>
<td>(Alternate</td>
<td></td>
</tr>
<tr>
<td>wrong answer)</td>
<td></td>
</tr>
<tr>
<td>UN text</td>
<td>The text is typed in response to an unanticipated</td>
</tr>
<tr>
<td>(Unanticipated</td>
<td>answer</td>
</tr>
<tr>
<td>answer)</td>
<td></td>
</tr>
<tr>
<td>TY text</td>
<td>The text is typed if prior conditions are met</td>
</tr>
<tr>
<td>(Type)</td>
<td></td>
</tr>
<tr>
<td>FP text</td>
<td>The text is typed and machine continues without</td>
</tr>
<tr>
<td>(Forward pointer)</td>
<td>waiting for a student response</td>
</tr>
</tbody>
</table>
where "label word" is a character string which names or labels the frame. Frames which are referenced by branch instructions must be labeled. A branch of the form

\[ \text{BR} \quad \text{label word} \]

causes the machine to branch to the frame labeled "label word."

The first frame in an instructional program is a definition and the subsequent frames are used either to question the student about the definition or to provide further information. This information may be additional definitions or statements or may be provided by calling other modules. The final frame is usually a forward pointer which suggests to the student topics which he may wish to consider next.

A frame which contains a definition consists of a DF command, which may or may not be preceded by a label. When the machine encounters a DF command in the program it types the associated text and waits for the student to respond. When the student responds, the machine proceeds to the next frame. The student's response is not evaluated, but is used only to signify that he is ready to continue, thus permitting the student to control his pace through the PI program.

A frame which contains a statement consists of an RD command. An RD command and a DF command are handled in exactly the same way when a program is executed. The DF command was introduced to facilitate the automatic generation of the script, and the distinction between these two commands will become clear when the procedure for producing instructional materials is given.

The most important extension of Coursewriter I is the introduction of a submodule call. Any module may call any other module as a
submodule, where the called module may also call any other module to arbitrary depths. This provides a capability whereby the instructional materials may be constructed in small units which can then be combined to produce larger units. Because the student may request any module including those which serve as submodules, he may select very specific units of PI material.

A frame which contains a submodule call consists of a CL command. A submodule call may be labeled. When the machine encounters a CL command, it suspends the current module and causes the called module to be administered. At the completion of the called module the module which was suspended is resumed with the frame following the submodule call.

Questions are used to test the student's understanding of the definitions. He may be quizzed about implications of the definition or about terms used in the definition.

A frame which contains a question consists of a QU command which may or may not be preceded by a label and must be followed by an answer. When the machine encounters a QU command in the program it types the associated text and waits for the student to respond. The answer contained in the instructional program is used to evaluate the student's response and thus determine the action to be taken. An answer consists of a correct answer and a wrong answer.

A correct answer consists of one or more correct answer sets. A correct answer set consists of a CA command which specifies the correct answer, zero or more CB commands which specify alternate forms of the correct answer, and an action to be taken if the student's response
matches the text in the CA command or any of the subsequent CB commands.

The action may be null, or it may consist of a type or TY command and/or a branch or BR command. If the action contains a TY command the machine types the associated text. If the action contains a BR command the machine branches to the specified frame. If the action does not contain a branch the machine then proceeds to the next frame.

Wrong answers are specified to provide differential action for various anticipated errors. A wrong answer consists of zero or more wrong answer sets followed by an unanticipated answer. A wrong answer set consists of a WA command which specifies a wrong answer, zero or more WB commands which specify alternate forms of the wrong answers, and an action to be taken if the student's response matches the text of the WA command or any of the WB commands. The action may consist of a TY command and/or a BR command which have the same effect as in the correct answer set. However, if the action does not contain a branch, the machine will remain in this frame and accept another response from the student.

A wrong answer must contain an unanticipated answer consisting of one or more UN commands each followed by an action. After executing a QU command, the machine will compare the student's response to all specified correct answers and then to all specified wrong answers until a match is found. If the student's response does not match any of the specified correct or wrong answers, the machine will type the text associated with the first UN command and then treat the action as it would the action for a wrong answer. Subsequent unanticipated answers
cause the UN commands to be executed in sequence, the last UN command being repeated if necessary.

A frame which contains a forward pointer consists of an FP command. An FP command is ignored whenever a module is executed as a submodule; otherwise the machine types the associated text but does not wait for a student response. This command was introduced to provide a means whereby a next topic may be suggested to a student. Because an FP command is normally the last frame in a module, the machine does not wait for a student response, thus the machine exits the routine which administers a PI module immediately upon completion of the final frame.

To summarize, a frame contains either a definition, a statement, a submodule call, a question or a forward pointer. If a frame contains a definition or statement, the machine types the statement and then waits for the student to respond. When the student responds the machine proceeds to the next frame regardless of the response from the student. If a frame contains a question, the machine types the question and waits for the student to respond. The student’s response is evaluated and the action to be taken next is a result of the instructional program and the student’s response. If the frame contains a submodule call, the called module is administered and then the machine proceeds to the subsequent frame. If the frame contains a forward pointer the machine types the associated text but does not wait for a student response.

There are two special responses, HELP and REPEAT which the student may enter in reply to a question. These requests, which are meaningful only in the tutorial mode, have the following effect.
If a student types HELP in response to a question, the machine will supply the correct answer, that is the answer specified by the first CA command in the frame, and then ask the student to repeat the answer. The machine then waits for another student response and proceeds as before.

If the student types REPEAT in response to a definition, statement or question, the machine will repeat the frame. The need for this command will become apparent when we consider the control commands which permit the student to suspend a module and return to it later. When the machine encounters the end of an instructional program, the machine exits the routine which administers a PI module and returns to the previous mode.

The Discussion Mode

The conversational routine for the discussion mode is based on the ELIZA program which was developed by Weizenbaum (Weizenbaum, 1966) and extended and programmed in GTL for the B-5500 by Fricks (Fricks, 1970). The data structure for the discussion mode, called a script, consists of three types of script elements: keyword elements, translate elements, and D-list elements. A description of the major functions of these elements is given below. A formal description of the script syntax and a script are given in Appendix C.

A script contains a number of decomposition rules, and associated with each decomposition rule is one or more recomposition rules. A decomposition rule is a template which must match the input string if the associated recomposition rules are to be utilized to generate a
response. The template consists of words which must be matched exactly by words in the input string; a "0" may be used to match any string of words including the null string. A decomposition rule together with its associated recomposition rules is called a composition rule.

When the student enters a statement or question, the discussion mode routine analyzes the input string and utilizes the composition rules to generate a response. Keywords are used to determine which composition rules to try. For each keyword there is a keyword element containing a priority and a list of composition rules. The input string is analyzed for the presence of keywords. If any keywords are present in the input string, the keyword with highest priority is selected and the corresponding decomposition rules are applied to the input string. If any of the decomposition rules matches the input string, one of the corresponding recomposition rules is utilized to generate a response. If no match is found, other keywords in the input string are tried until a matching recomposition rule is encountered. A special procedure is provided to generate a response when none of the recomposition rules in the script match the input string.

The analysis proceeds as follows. The input string is scanned from left to right. The first keyword encountered is placed on a keyword stack. Subsequent keywords in the string are placed on top of the keyword stack if they have higher priority than the keyword presently on top of the stack, otherwise they are placed on the bottom of the stack. When the scan of the input string is completed the keyword with highest priority will be on top of the stack. The order of other
keywords within the stack depends upon both their priority and their sequence in the input string, however keywords of equal priority will occur in the stack in the order in which they occurred in the input string.

If any keywords were detected in the input string, the keyword at the top of the stack is selected. If any of the decomposition rules for the selected keyword match the input string, one of the recomposition rules associated with that decomposition rule is selected to generate the machine response. If none of the decomposition rules matches the input string, this process is repeated using the next keyword from the keyword stack. When certain conditions are met, keywords cause an additional reply to be generated and stored in the memory stack for later use.

If no keywords are encountered in the input string, the routine will either use one of the responses stored in the memory stack or a response will be generated by selecting a recomposition rule from the recomposition rules given for the special pseudo-keyword "NONE." If keywords are encountered but all keywords are tried without yielding a matching decomposition rule, then one of the recomposition rules for the special pseudo-keyword "NOHIT" is used to generate a response.

A keyword element has the form

\[(K \ p \ ((D_1)(R_{1,1})(R_{1,2})...(R_{1,m_1}))
\]

\[((D_2)(R_{2,1})(R_{2,2})...(R_{2,m_2}))
\]

\[...
\]

\[((D_n)(R_{n,1})(R_{n,2})...(R_{n,m_n}))\]
where \( K \) is a keyword, \( P \) the priority and

\[
((D_1)(R_{i,1})(R_{i,2})...(R_{i,n_i}))
\]

is a composition rule in which \( D_1 \) is a decomposition rule and \( R_{i,1} \) through \( R_{i,n_i} \) are the recomposition rules for \( D_1 \). Thus a keyword element consists of a keyword, a priority and a set of composition rules.

An example of a keyword element is as follows:

\[
(\text{OPERATIONS 10 ((0 LINEAR LISTS 0)}
\]

\[
(\text{OPERATIONS ON LINEAR LISTS INCLUDE}
\]

\[
\text{ACCESSING; INSERTING AND DELETING NODES %})}
\]

If the student enters the question

\[
\text{WHAT ARE OPERATIONS ON LINEAR LISTS}
\]

this keyword element would cause the machine to reply

\[
\text{OPERATIONS ON LINEAR LISTS INCLUDE}
\]

\[
\text{ACCESSING, INSERTING AND DELETING NODES.}
\]

It is not possible to use periods, commas, or question marks in a recomposition rule because these symbols have special meaning to the computer system. Therefore, the letter \( Q \) is used as a question mark designator, a semi-colon is used as a comma designator, and a per cent sign is used as a period designator. The appropriate transformation is applied before the response is printed.

Alternate forms of the keyword element are provided to simplify preparation of a script by avoiding redundant recomposition rules.

It is sometimes desirable to use the same set of composition
rules for more than one keyword. This is accomplished by using a special form of the keyword element which contains a reference element. The keyword element then takes the form

(keyword P (=newkeyword))

which causes the composition rules for "newkeyword" to be used for "keyword."

A keyword element may also include a keyword equate part which causes all occurrences of the keyword in the input string to be replaced by another word prior to the decomposition process. A keyword element with a keyword equate part takes the form

(keyword P = newword...)

where keyword is to be translated to newword. This translation occurs independently of which keyword is selected.

A reference element and a keyword equate part are often used together. For example, consider the keyword element given above for "OPERATIONS." It may be desirable to treat "OPERATION" as a keyword with the same composition rules. This may be accomplished by using the following keyword element

(OPERATION 10 = OPERATIONS ((=OPERATIONS)))

If the student enters the question

WHAT IS AN OPERATION ON LINEAR LISTS

the machine response may be

OPERATIONS ON LINEAR LISTS INCLUDE
ACCESSING, INSERTING AND DELETING NODES.

Because the same text is frequently needed in both a PI module
and the script, a frame reference has been introduced. A frame refer-
ence, which is used as an element in a recomposition rule, specifies a
module name and frame number. The machine generated response is then
the text of the specified frame. This does not cause a shift to tuto-
rial mode.

In certain instances, the appropriate response to a statement or
question from the student may be an entire instructional program. In
this case, a module reference, which specifies the module name, is used
instead of recomposition rule. This causes the machine either to shift
to tutorial mode and administer the specified module or to print the
text of the first frame depending upon whether or not locus\(^2\) of control
is with the machine.

The keyword element is thus used to specify which words are to
be used as keywords, to assign priorities to keywords, to specify
transformations on keywords and to associate with each keyword a list
of composition rules.

A translate element is used to specify a transformation to be
performed on a word which is not a keyword. A translate element has
the form

\[ \text{(word = newword)} \]

and causes "word" to be replaced by "newword."

For example, we might wish to treat "LINEAR LIST" and "LINEAR
LISTS" in the same way. If neither "LIST" nor "LISTS" is a keyword, we
would use the translate element

\[ \text{(LIST = LISTS)} \]
thus causing "LIST" to be replaced by "LISTS." This would cause the
input string

OPERATIONS ON A LINEAR LIST
to be matched by the decomposition rule for "OPERATION" in the above
example.

The third type of script element is a D-list element. The
primary use of the D-list element in the current version of the system
is in connection with the memory stack. If the script contains a D-list
element

(keyword / MEM)

then the memory mechanism will be activated whenever "keyword" is the
keyword in the input string which is selected for processing. Prior to
generating a response for immediate use, an attempt will be made to
match a decomposition rule associated with the pseudo-keyword "MEMRULE"
with the input string. If a match occurs a response will be generated
according to the corresponding recomposition rules and stored for later
use.

A procedure for generating a script is given in Chapter VII.

Commands for Shifting Modes

GITIT is initially in the discussion mode, with the automatic
shift to tutorial mode disabled. Special control commands are provided
to permit the student to change the mode in which the system is oper­
ating.

All control commands begin with an asterisk (*). They may be
entered at any time because any response from the student which begins
with an * is handled by a special control routine rather than by the routines for the discussion or tutorial mode.

There are three commands used to shift modes:

*DISCUSS causes the machine to enter the discussion mode;
*TUTOR causes the machine to enter the tutorial mode;
*RETURN causes the machine to return to the previous mode.

If the student wishes to be tutored, he can type "*TUTOR modulename" to select a module. He may then enter "*TUTOR," "*DISCUSS," or "*RETURN" whenever he wishes to shift modes again.

There appear to be conditions in which an automatic shift to tutorial mode is useful. The pair of commands

*CONTROL ON
and  *CONTROL OFF
have been provided to enable and disable the machine-controlled shift to tutorial mode.

If control is on and the student asks a question or makes a statement about a topic which is the main subject of a PI module, the machine will automatically shift to tutorial mode and begin to administer the module. The student may type *RETURN to cause the machine to return to discussion mode thereby rejecting the module, otherwise the machine will return to discussion mode at the completion of the module.

If the student asks about a more specific topic which is discussed in a PI module but not the subject of an entire module, the machine will generate a response together with a reference to the PI module, but it will remain in discussion mode. The student may cause
the machine to shift to tutorial mode and administer the PI module if he wishes to do so.

Several other special control commands are provided, including

*HELP

which causes the machine to administer a special PI module which explains the control commands,

*START OVER

which restarts the system in the discussion mode eliminating all of the suspended processes, and

*STOP

which terminates the computer program GITIT.

Summary

GITIT is an experimental CAI system which operates in either the discussion or the tutorial mode. When the system is in discussion mode the student controls the order of presentation of materials, thus locus of control is with the student. When the system is in tutorial mode the order of presentation of material is controlled by the machine under the control of an instructional program, therefore locus of control is with the machine.

The system can shift from discussion mode to tutorial mode and back, thus permitting locus of control to pass from the student to the machine and back to the student. Either the machine or the student can instigate a shift of mode.

The data structures for the two modes reflect the difference in information requirements for machine control and student control.
The data structure for the tutorial mode must provide the information necessary to control the order of presentation of material. Therefore, an instructional program is used in which the sequence of the frames controls the order of presentation of the material contained in the frames. Since it is not necessary to recognize many different responses from students for a given question, the anticipated responses are included in the program eliminating the need for elaborate recognition procedures.

The script which is the data structure for the discussion mode does not contain any information on the order of presentation of materials because the student controls this order. However, the script does contain provision for elaborate decomposition and recomposition rules because it is necessary to recognize a large number of different statements and questions at any time.

The data structures for both tutorial and discussion mode may contain the information required to initiate the execution of a PI module.

The combined data structure for the two modes, which consists of a collection of PI modules and a script, is called a local data structure. Chapter VII provides a procedure for generating a local data structure.
A local data structure for the dual-mode CAI system GITIT consists of a set of related PI modules and a script. The script supports discussion of the material contained in the PI modules and serves as an index to the modules. Associated with each set of PI modules is a specified technical vocabulary. The objective of the following procedure is to produce a set of PI modules and the specified technical vocabulary, and to utilize the vocabulary and modules to automatically generate a script.

Although the present version of GITIT is designed to operate on one local data structure at a time, the design permits extension of the system to operate on a global data structure. The proposed extension to global data structures is discussed in Chapter VIII. The following procedure generates a local data structure suitable for inclusion in a global data structure.

This procedure is designed to generate instructional materials which teach a set of consistent verbal definitions. The accommodation of examples, applications, or other types of material is not considered at this time.
Development of Definitions

The first step is to determine the major terms whose definitions are to be taught, and to develop a verbal definition (or description) for each of these terms, where "term" is used here to mean the word or phrase being defined. The technical vocabulary will include all of the terms defined as well as the technical terms used in their definitions. All terms used in definitions must either be stated to be primitive terms, that is, terms whose definitions are prerequisite to the unit, or must themselves be defined. Therefore it is necessary to determine the terms which are used in the definitions and to augment the set of definitions as required to assure that all terms not declared to be primitive have been defined. If new terminology is introduced in these definitions the process is repeated until definitions have been given for all terms which are not primitive. Care must be taken to assure that all terminology is consistently used.

The determination of the terminology used in the definitions may be facilitated by writing all of the technical terms used in a definition beneath the definition as shown below:

A linear list is a set of \( N \geq 0 \) nodes, \( X[1], X[2], \ldots, X[N] \) whose structural properties involve only the linear order of the nodes.

```
set
node
linear order
```

Judgment must be exercised to select only those terms which should be considered a part of the technical vocabulary for the area. In the above example, "node" is clearly being used in a technical sense.
and is subject to definition within the area, "set" and "linear order" are primitive terms, and "involve" is not considered here to be used in a technical sense. "Structural properties" is questionable; its meaning is somewhat apparent from its usage. It may be useful to regard the terms as prerequisites to the understanding of the definition, that is, terms which the student will need to know in order to understand the given definition.

When all terms in the technical vocabulary have either been defined or stated to be primitive, the actual preparation of the instructional materials can proceed.

Writing the PI Modules

One PI module should be developed for each definition and each module should take the following form.

Frame one gives a statement of the definition and an indication of the terms being used in the definition. For example, the first frame of the module on linear lists is:

```
DF  A LINEAR LIST IS A SET OF N GREATER THAN OR EQUAL TO ZERO NODES X[1],X[2],...,X[N] WHOSE STRUCTURAL PROPERTIES INVOLVE ONLY THE LINEAR ORDER OF THE NODES.
BP  *SET *NODE *LINEAR ORDER
```

A BP command is used to list the terms used in the definition where each term is preceded by an asterisk. These terms will be called backward pointers. In this example, "set" and "linear order" are primitives and "node" is defined in another module although the distinction between primitives and terms defined in other modules is not shown in the frame.
Subsequent frames in the module are used to test the student's understanding of the definition or to give further information. The testing may use a paraphrase of the definition or questions concerning terms used in the definition. These frames may introduce subdefinitions and may involve additional terms. Several possible frames in a module on linear lists are given below as examples.

Example 1: Paraphrase of Definition

**Question:** WHICH IS THE MOST PRECISE DEFINITION OF A LINEAR LIST?

**Answer:** A LINEAR LIST IS...
1. A FINITE SET OF NODES.
2. A TOTALLY ORDERED SET OF NODES.
3. A (POSSIBLY EMPTY) FINITE SET OF NODES WHICH IS TOTALLY ORDERED.

**Response:**
- BP *FINITE SET *TOTALLY ORDERED SET *EMPTY SET
- CA 2
- TY IF YOU ARE THINKING OF FINITE SETS, YOU ARE RIGHT.
- CA 3
- TY CORRECT
- WA 1
- TY NO - THE LINEAR STRUCTURE IS NOT REPRESENTED. TRY AGAIN.
- UN 1, 2, OR 3?

Example 2: Comment on Preceding Frame

**Response:** NUMBER 3 IS THE BEST CHOICE BECAUSE IT SPECIFIES THAT THE SET MUST BE FINITE AND TOTALLY ORDERED, AND THAT IT MAY BE EMPTY, NUMBER 2 IS ALMOST CORRECT, BUT FAILS TO SPECIFY THAT THE SET IS FINITE.

Example 3: Testing Understanding of Definition

**Question:** CAN A LINEAR LIST BE EMPTY, THAT IS CAN A LINEAR LIST CONTAIN ZERO NODES?

**Answer:**
- CA YES
- TY RIGHT
- WA NO
- TY REREAD THE DEFINITION AND TRY AGAIN.
- UN PLEASE ANSWER YES OR NO.
Example 4: Additional Information

DF FOR N GREATER THAN ZERO, X[1] IS THE FIRST NODE.
FOR K GREATER THAN 1, X[K] IS PRECEDED BY X[K-1].
FOR K LESS THAN N, X[K] IS FOLLOWED BY X[K+1].
X[N] IS THE LAST NODE.

There are, of course, many possible questions and there is no
specific rule as to which to use. The general idea is to provide ade­
quate questions to permit the student to determine whether or not he
understands the definition.

There will be one PI module for each term which is defined.
Therefore one PI module should be produced for each definition generated
by the previous step. It may be the case that one definition is a sub­
definition of another. For example, accessing, inserting and deleting
are operations which can be performed on a linear list. Because a
module can contain submodules, this situation is handled by producing
modules on accessing, inserting and deleting and then producing a module
on operations which contains these three modules as submodules.

In the process of generating the PI modules, additional termi­
nology will arise. Technical terms should be added to the technical
vocabulary as they are used. Sometimes terms will arise which should
be defined in a submodule as discussed above. A rule of thumb is to
consider a submodule whenever two or more frames are devoted to a sub­
definition.

A full explanation of the commands used in the PI modules is
given in Chapter VI, therefore the present discussion will be limited to
the way in which the commands are to be used.
The first frame, which is the definition of the major term, must be a DF command. In subsequent frames a DF command should be used for statements which should be considered as a possible response in discussion mode and an RD command for other statements. An RD command will be treated in precisely the same manner as a DF command when the PI module is executed, but the information contained in RD commands will not be placed in the script. All commands other than DF commands are ignored when the script is generated therefore information which is to be available in the discussion mode must be placed in a DF command.

The final frame in each module may be used as a forward pointer (FP command), which provides a reference to subsequent material. An appropriate forward pointer will be generated by a subsequent step, therefore the FP command is not used in initially writing the modules.

It is imperative that careful attention be given to the specification of the technical vocabulary which is used in the PI modules. Only by carefully controlling this vocabulary and keeping the usage of terminology completely consistent within the modules can an effective supporting script be generated.

The technical vocabulary will consist of three different types of terms: major terms which have a defining module, minor terms which are defined within another module, and primitive terms the definitions of which are prerequisite to the subject area.

After the PI modules have been generated, they should be reviewed to be sure that all technical terminology has been entered in the technical vocabulary. Then a dependence matrix should be prepared in which
there is one row and one column for each major term and one additional column for listing minor terms or primitive terms. The row for each major term should be completed as follows:

1. Enter I in the column of each of the other major terms used in the definition of this major term.

2. Enter II in the column of each of the other major terms used in the module but not in the definition of the major term.

3. Enter S/M in the column of each of the other major terms which are included in the module by a CL command calling the module defining that term.

4. Enter III in the column of each of the other major terms which are used in a submodule called by this module.

This matrix is used to generate the forward pointers for the module. Figure 31 shows the dependence matrix for the experimental materials.

The row for term B will contain a I in the column for term A whenever Term A is used in the definition of term B. In this case the module defining term A may contain a forward pointer to the module defining term B. The forward pointer may be the sentence

A TOPIC TO CONSIDER NOW IS term B.

or any other statement that will serve to suggest term B to the student.

When a forward pointer to term B is placed in module A, an upward pointing arrow (↑) should be placed in row A column B. It is significant that both forward and backward pointers are terms rather than module names because this means that a module can be modified or additional modules generated without having to change other modules provided
Figure 31. The Dependence Matrix for Experimental Materials
the technical vocabulary is changed as required.

A dependence graph of the modules within an area indicating forward and backward pointers may be drawn from the matrix. The matrix for the experimental materials was shown in Figure 31 and the corresponding graph is shown in Figure 32.

The responses which are generated by the machine in discussion mode are extracted from DF entries in the PI modules for both major and minor terms. In order to introduce replies to questions about primitive terms, special PI modules must be written. A reply concerning a primitive term should tell the student that the term he asked about is a primitive in the current area, and should help him locate the information he needs outside this area. When multiple local data structures are available on different subjects, he should be provided with a term which will be recognized by the global script to direct him to the appropriate area. For example, the response to the primitive term

WORDS OF COMPUTER MEMORY

might be

COMPUTER CONCEPTS SUCH AS WORDS OF COMPUTER MEMORY ARE PREREQUISITE TO THIS UNIT.

The student could then enter the global index with "computer concepts" and be directed to an appropriate local data structure if such is available.

This is accomplished by introducing "computer concepts" as a major term and writing a PI module containing an initial frame on computer concepts and additional frames, all DF commands, which give statements about primitive terms from the computer concepts area.
Figure 32. The Dependence Graph for Experimental Materials
One such module should be prepared for each prerequisite area. One of these modules can be selected by the student if he wishes to determine what he is expected to know from a specific prerequisite area.

Because a global script is not available in the current implementation, references to reading assignments have been provided for primitive terms. Subjects reported that a chapter reference in a book was more satisfying than a statement that the term in question was primitive, even when they had no intention of reading the chapter.

The terms in the technical vocabulary may consist of words or phrases, but they must not include special characters. For purposes of this procedure a word is a string of at most 31 upper case alphabetic characters and numerals, the first of which must be alphabetic. Words may not be hyphenated. Therefore, "LINEAR LIST" is an allowable phrase consisting of the two words "LINEAR" and "LIST"; "LINEAR-LIST" is not acceptable as a term. The word "CASE1" or the phrase "CASE ONE" are acceptable as terms, but "CASE 1," "CASE-1" and "CASE-ONE" are not acceptable. The technical vocabulary should be edited to assure that the preceding rules are followed.

Each module must have a unique module name which is a word as defined above except that it must not exceed six characters in length.

Automatically Generating the Script

The script, which is the portion of the data structure used in discussion mode, must be able to recognize the terms in the vocabulary. The recognition scheme is based on keywords which are words as defined above rather than phrases, therefore decomposition rules are required.
to recognize phrases. The first step in generating the script is to generate the decomposition rules from the vocabulary.

In preparing the vocabulary for machine processing it is necessary to specify alternative forms of a term. For example, "LINEAR LIST" and "LINEAR LISTS" are two entirely different phrases to a computer unless the computer is specifically told otherwise. Therefore a facility has been provided to specify that certain words are to be translated to other words or phrases before the decomposition process begins. A list must be made of all words which are to be translated to a new word or phrase prior to the decomposition process. A translate element will be placed in the script for each word to be translated.

It may also be necessary to specify alternative forms of a major term. For example, "link," "pointer" and "address of a node" are synonyms and therefore are defined by the same module. It may also be desirable to specify "node address" as a synonymous major term. Prepositions, articles and conjunctions will be ignored, therefore "address of node" and "address of a node" will be treated as the same term, as will "address node."

Algorithm A which is given in Appendix E will generate a script to recognize all terms in the technical vocabulary. A one-word term will be a keyword. For a multi-word term, the first word in the term will usually be a keyword and the other words, excluding prepositions, articles and conjunctions, will occur in a decomposition rule. The same word may, of course, be the keyword for more than one term, therefore the decomposition rules are necessary to differentiate between terms.
Each keyword in the script has a priority. Whenever two or more keywords appear in the same student input, the keyword with highest priority is considered first. The decomposition rules for that keyword are applied to the input and the first decomposition rule which matches the input is selected. If none of the decomposition rules match, another keyword is selected. Therefore both the priority of the keywords and the order of the composition rules for a given keyword affect which of multiple terms is utilized to generate a reply.

An attempt is made to provide the most specific response which is appropriate to the student input, therefore minor terms are given precedence over major terms in the recognition process whenever this is possible. Similarly major terms are given precedence over primitive terms. The order of decomposition rules for a given keyword is controlled by processing minor terms first, then major terms and finally primitive terms. Because new rules are added at the end of the list, the first rules generated will be the first rules tried in the decomposition process. The only exception is that each decomposition rule is placed before any other decomposition rules which would match the same term.

A keyword may serve as the keyword for several terms of different types, therefore it is not always possible to apply different priorities to the different types of terms. The priority of a keyword is assigned according to the type of term in which it first occurs as a keyword, using priorities of 10, 20 and 30 for primitive terms, major terms and minor terms, respectively. Thus a keyword occurring in minor
terms is given the highest priority, a keyword in either major or primitive terms but not in minor terms is given a lower priority, and a keyword which occurs only in primitive terms is given lowest priority.

Care must be taken in determining which word in a multi-word phrase is to serve as a keyword to assure that the decomposition process will recognize the entire phrase. For example, assume that we wish to recognize the term "pointer field" and we have selected "field" to be the keyword. However, if "pointer" is also a keyword and has a higher priority than "field," or if "pointer" and "field" have equal priority and "pointer" is scanned first in analyzing the student input, the term "pointer" rather than "pointer field" may be recognized by the decomposition process. Similarly, if "field" is the keyword, "field" rather than "pointer field" may be recognized. The order in which the decomposition rules are tried must be such that the decomposition rule for "pointer field" is encountered prior to the rule for "pointer" or the rule for "field."

Decomposition rule $r_1$ will be said to cover decomposition rule $r_2$ just in case that any string matched by rule $r_2$ is also matched by rule $r_1$. This introduces a partial order on the set of decomposition rules, and the order in which the decomposition rules are tried for any input string must preserve this partial order.

The order in which decomposition rules are tried is determined by the priority of keywords, the sequence of keywords within the input string, and the order of decomposition rules in the list for a given keyword. Algorithm A utilizes this partial order in assigning
priorities to keywords and generating decomposition rules to assure that
the order in which the rules are tried respects the partial order.

In the resulting script, all single word terms are keywords with
the decomposition rule "(0)" and a priority of 10, 20 and 30 for primi-
tive terms, major terms, and minor terms, respectively. For multi-word
terms, the first word of the term is the keyword with priority as stated
above unless another word in the term is a keyword with higher priority
in which case it is taken as the keyword for this term. Because the
keyword selected from the student input is the keyword with highest
priority or the first of several words of equal priority, this assures
that every term in the vocabulary can be recognized.

Script elements for the special keywords "NONE," "NOHIT" and
"MEMRULE" are also generated. The use of these keywords, as explained
in Chapter VI, is to provide a response when none of the decomposition
rules match the student's input. D-list elements of the form

(word/MEM)

are placed in the script for each word which is a keyword for a major
term, and appropriate decomposition rules are placed in the list of
composition rules for the special keyword MEMRULE.

Algorithm A will produce a script with decomposition rules to
recognize all terms in the technical vocabulary. Algorithm B, also
given in Appendix E, will insert appropriate recomposition rules based
on the DF commands in the PI modules. The procedure for each PI module
is as follows.

A module designator of the form
is inserted in the script as a recomposition rule for major terms.

Then each DF command other than the one in the first frame is analyzed for the presence of minor terms which are not backward pointers for that frame. A recomposition rule containing the frame designator

is inserted as a recomposition rule for all minor terms other than backward pointers which are contained in the DF command beginning in record n of module modulename.

If the module contains a forward pointer the frame reference

is placed with the decomposition rule for the major term under the special keyword MEMRULE, where "n" is the record number of the FP command.

For each major term, a recomposition rule to generate the reply

is also placed in the list of rules for the special keyword NONE.

Primitive terms are processed in a similar manner, using the special PI modules for the prerequisite areas.

Finally, recomposition rules to generate the statements

are placed under the special keyword NOHIT.

Thus Algorithm B will insert recomposition rules in the script.
based on DF commands in the PI modules. If the same minor or primitive term occurs in the text of more than one DF command there will be multiple recomposition rules for that term and one such rule will be selected whenever the composition rule is used. It may also be the case that no recomposition rules will be generated for some terms. This may be the result of an error in specifying the vocabulary or the omission of material using that term. This situation should be investigated and the necessary corrections made to the PI modules or the technical vocabulary. A new corrected script can then be generated.

The script presently in use was generated manually using the algorithms given in the Appendix. However, the algorithms are completely specified and do not require any human judgment. Therefore, a computer program can be produced which will use the modules and the vocabulary together with an indication of the module name for each major term to generate the script. This will further simplify the preparation and subsequent modification of instructional materials.

**Suggested Variations in the Algorithms for Generating the Script**

There are many variations of the algorithms which could be tried and investigated empirically to determine the effect on the operation of the system.

In generating the decomposition rules, the reduced term

\[ 0 \text{WD}_1 0 \text{WD}_2 0 \ldots 0 \text{WD}_n 0 \]

is used where a 0 is inserted between each two consecutive words. Thus the decomposition rule may fit a different phrase in which the
corresponding words happen to occur in the same order. The possibility that this will happen could be reduced by inserting a zero in the term only where a conjunction, preposition or article has been deleted. This should cause the recomposition process to more precisely recognize the technical terms, but it might also fail to recognize an unspecified variation of the term.

The first frame of a PI module is used as a recomposition rule only for the major term and not for minor terms occurring in the text. It might be desirable to use the definition of major terms in the script so that they may occur as responses to minor terms contained in the definition.

In determining which minor terms and primitives are contained in the text of a DF entry, terms used as backward pointers are excluded. Thus the text of a DF entry would never be produced in response to any of the terms which are specified to be backward pointers. Perhaps it would be useful to use this text which would actually serve as a forward pointer.

Whenever the first frame of a module contains a major term as a backward pointer, a recomposition rule to generate the reply

\[ \text{HAVING DISCUSSED prevterm, PERHAPS WE} \]

\[ \text{SHOULD CONSIDER term} \]

could be placed with the decomposition rule for prevterm associated with the keyword MEMRULE, where term is the major term for the current module and prevterm is the backward pointer. This would provide additional forward pointers.
Variety could be introduced by providing a list of recomposition rules such as

(LETS DISCUSS term%)

(SHALL WE TALK ABOUT term Q)

and selecting one at random for inclusion in the list of recomposition rules for the keyword NONE. This might have the effect of raising the student's interest level by increasing the novelty of the situation.

Summary

The generation of instructional materials involves three steps:
1. Develop the definitions of the major terms.
2. Write one PI module for each major term.
3. Generate a script to support discussion of the material contained in the PI modules.

The effectiveness of the instructional materials is dependent upon an accurate specification of the technical vocabulary for the area and the consistent use of this vocabulary within the area.
CHAPTER VIII
CONCLUSIONS AND RECOMMENDATIONS

The conclusions presented here have all been discussed in the preceding chapters, therefore they are presented without further discussion or explanation. Attention is then given to recommendations for further research.

Conclusions

The major conclusions of this research based on the objectives set forth in Chapter I are as follows:

1. The qualitative model of instructional control developed herein provides a framework for describing and comparing existing CAI systems with respect to control issues.

2. The experimental CAI system which was developed includes a data structure which (a) supports a dual-mode CAI System, i.e. one in which either the student or the computer may control the information flow, (b) has a structure of sufficient complexity to support a wide range of learning tasks, i.e. rote learning, learning concepts in their interrelationship, browsing, etc., and (c) efficiently utilizes a hierarchical arrangement of computer memory such that there is an effective match between performance requirements and the speed and cost of memory.

3. The present data structure is adequate to support the free
learning activities of a student who is able to validly evaluate his own performance, but additional structure would be required to provide both more feedback to the student and extended machine control for students incapable of valid self-evaluation.

4. Student performance on the system confirmed expectations based on prior research as to variability of individual strategies and effectiveness in a complex learning task.

5. Valid self-evaluation is important if a student is to succeed in a free-learning task.

6. Locus of information control should be with the student when he has an appropriate strategy for the task; locus of information control should be with the computer when the student does not have an appropriate strategy.

7. The system permits the display of a wide range of learning strategies.

8. The system is adequate to support a large variety of learning tasks.

Additional results of the research are given below:

1. The conceptual model of control served as a guide to the design of a CAI system with respect to (a) the following types of control: immediate control, control of a communication event, information control, procedural control, and the control of the topic area, (b) the commands required for each type of control, and (c) the data structure required to support such a system.

2. A well-specified procedure was developed for generating the
instructional materials required to teach an interrelated set of consistent definitions.

3. An algorithm is presented for developing a script to support student-computer discussion of the subject content of instructional modules.

4. Due to the modular design of the instructional materials and the fact that the links between modules are algorithmically generated, additions and modifications may be made without change to other modules.

5. The computer program for the CAI system is completely subject-matter independent which permits the system to be used for the investigation of a wide range of instructional problems.

6. The final CAI system proved to be efficient with regard to (a) the utilization of computer time, (b) the utilization of computer memory, and (c) the preparation of instructional materials. Therefore the experimental CAI system is feasible for use in practical application.

Recommendations for Further Research

Due to the efficiency of GITIT in its present form in terms of generation of instructional material and computer utilization, GITIT appears to have a range of immediate application. Such applications would suggest the extension of the system to operate on a global data structure. Furthermore the system has potential application in a number of areas which have not been explored in the present dissertation; for example, information acquisition, support for review writing,
information storage and retrieval, etc. The present set of recommenda-
tions is limited to the use of GITIT as a method for investigating man-
machine interaction with special reference to teaching and learning
strategies.

Programming the Script Generation Algorithm

The algorithm for generating a script which was presented here
is completely specified and suitable for execution by a computer. How­
ever, it has not yet been programmed and the manual generation of
scripts using the algorithm is tedious and prone to human error because
of the clerical detail involved. This algorithm should be programmed
and then tested on a variety of materials.

A Study of Definitions

The nature of relationships involved in definitions and the rela­
tionships between interrelated items which form a chain of definition
need to be studied at a more analytic level with the aim of specifying
even further the method of generating consistent sets of definitions.
This would facilitate the preparation of instruction modules and provide
further guidelines for specifying terms to be included in the technical
vocabulary. It may also suggest improvements in the algorithm for
generating the script.

Investigation of a Range of Tasks

The present system can support a wide range of tasks. Consider­
ation should be given to the various tasks which can be supported and a
variety of these tasks investigated. Attention should be given to
tasks requiring forward pointers such as browsing and learning the
relationships between concepts.

Effects of Feedback on Student Performance

The effects of feedback on student performance should be systematically investigated. This investigation should include a consideration of the effectiveness of a comparison of concepts in terms of similarities and differences as a test of the adequacy of learned definitions as well as an example checking device to test the application of definitions.

The Use of GITIT to Teach Students How to Learn

Strategy teaching procedures should be incorporated into the system. The system could then be used to investigate the effects of these procedures on locus of control and the possible development of general learning strategies by students.

Pask (Pask, 1972) has demonstrated that students can be taught to learn and that the effect of learning to learn generalizes over various aspects of the same subject area. Because of the subject-matter independence of the GITIT system and the ease with which instructional materials can be prepared for different topic areas, GITIT can be utilized to study the generalization of learning to learn across subject areas.

Pask contends that some students are serialists and others are holists and that teaching a student how to learn consists of strengthening a student's ability to use his natural strategy. But it may be the case that a serialist strategy is appropriate for some tasks and a holist strategy is appropriate for other tasks. If so, it would appear
that students should be taught to use both strategies and to select the appropriate strategy for a given task. This possibility should be investigated.

**Local and Global Data Structures**

The design concept of GITIT permits the integration of local data structures containing information about different but related topic areas into a global data structure. The extension of the system to operate on a global data structure would permit investigation of questions concerning the division of subject matter material into topic areas and an empirical study of methods of determining when topic areas have been exhausted and under what circumstances new topic areas should be investigated.
APPENDICES
APPENDIX A

INSTRUCTIONAL MATERIALS FOR THE CASE STUDY

The instructional materials for the case study were prepared according to the prescribed method given in Chapter VII. This appendix includes a listing of the resulting vocabulary, the PI modules, and the script. The subject content of these materials was adapted from the second chapter of Knuth (Knuth, 1969).

The Vocabulary

The vocabulary consists of major terms, minor terms and primitive terms. Table 17 gives the major term and all specified alternate forms of the major term for each of the 15 modules. The keyword in each item is italicized.

The minor terms and primitive terms are listed in Tables 18 and 19, respectively, and the keyword in each term is italicized. The module name and record number of the response or responses generated for these terms are also given in these tables. The primitive terms "COMPUTER CONCEPTS" and "SET THEORY" do not have record numbers because they name prerequisite areas and thus refer to the entire module.

As explained in Chapter VII, it is sometimes desirable to perform certain transformations prior to the decomposition process. A list of the words which are to be translated and the resulting word or phrase are given in Table 20.
Table 17. Major Terms for Each Module in Experimental Materials

<table>
<thead>
<tr>
<th>Module Name</th>
<th>Major Terms for the Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>NODE</td>
<td>node</td>
</tr>
<tr>
<td>FIELD</td>
<td>field</td>
</tr>
<tr>
<td>LINK</td>
<td>address of a node</td>
</tr>
<tr>
<td></td>
<td>link</td>
</tr>
<tr>
<td></td>
<td>pointer</td>
</tr>
<tr>
<td>LLIST</td>
<td>linear list</td>
</tr>
<tr>
<td></td>
<td>list</td>
</tr>
<tr>
<td>LINKF</td>
<td>link field</td>
</tr>
<tr>
<td></td>
<td>pointer field</td>
</tr>
<tr>
<td>ALLOC</td>
<td>storage allocation</td>
</tr>
<tr>
<td></td>
<td>allocation</td>
</tr>
<tr>
<td></td>
<td>linked and sequential allocation</td>
</tr>
<tr>
<td>SALLOC</td>
<td>sequential allocation</td>
</tr>
<tr>
<td>LALLOC</td>
<td>linked allocation</td>
</tr>
<tr>
<td>OPERNS</td>
<td>operations</td>
</tr>
<tr>
<td></td>
<td>accessing and inserting</td>
</tr>
<tr>
<td></td>
<td>accessing and deleting</td>
</tr>
<tr>
<td></td>
<td>inserting and deleting</td>
</tr>
<tr>
<td>ACCESS</td>
<td>access</td>
</tr>
<tr>
<td>INSERT</td>
<td>insert</td>
</tr>
<tr>
<td>DELETE</td>
<td>delete</td>
</tr>
<tr>
<td>STKQUE</td>
<td>stacks and queues</td>
</tr>
<tr>
<td>STACK</td>
<td>stack</td>
</tr>
<tr>
<td>QUEUE</td>
<td>queue</td>
</tr>
</tbody>
</table>
Table 18. Minor Terms in Experimental Materials

<table>
<thead>
<tr>
<th>Term</th>
<th>Module</th>
<th>Record</th>
<th>Term</th>
<th>Module</th>
<th>Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>access in the middle of a stack</td>
<td>STACK</td>
<td>20</td>
<td>insert after a node</td>
<td>INSERT</td>
<td>17</td>
</tr>
<tr>
<td>address of a field</td>
<td>LINK</td>
<td>25</td>
<td>insert at the end</td>
<td>INSERT</td>
<td>4</td>
</tr>
<tr>
<td>bead</td>
<td>NODE</td>
<td>4</td>
<td>insert in the middle</td>
<td>INSERT</td>
<td>9</td>
</tr>
<tr>
<td>bottom of a stack</td>
<td>STACK</td>
<td>9</td>
<td>item</td>
<td>NODE</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>ALLOC</td>
<td>4</td>
<td>last node</td>
<td>LLIST</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>SALLOC</td>
<td>6</td>
<td>linked list</td>
<td>LLIST</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LLIST</td>
<td>47</td>
</tr>
<tr>
<td>content of a field</td>
<td>NODE</td>
<td>89</td>
<td>lifo</td>
<td>STACK</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>INSERT</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>content of a node</td>
<td>DELETE</td>
<td>9</td>
<td>linked list</td>
<td>LALLOC</td>
<td>5</td>
</tr>
<tr>
<td>delete at the end</td>
<td>DELETE</td>
<td>4</td>
<td>10</td>
<td>SALLOC</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>9</td>
<td>loc</td>
<td>SALLOC</td>
<td>16</td>
</tr>
<tr>
<td>delete in the middle</td>
<td>DELETE</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>9</td>
<td>locate</td>
<td>ACCESS</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dummy field</td>
<td>NODE</td>
<td>68</td>
<td>name of a field</td>
<td>NODE</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>LALLOC</td>
<td>38</td>
<td>name of a node</td>
<td>NODE</td>
<td>89</td>
</tr>
<tr>
<td>empty list</td>
<td>LLIST</td>
<td>36</td>
<td>pop</td>
<td>STACK</td>
<td>40</td>
</tr>
<tr>
<td>empty node</td>
<td>NODE</td>
<td>4</td>
<td>preceding node</td>
<td>LLIST</td>
<td>25</td>
</tr>
<tr>
<td>end of a list</td>
<td>STACK</td>
<td>9</td>
<td>push down</td>
<td>STACK</td>
<td>36</td>
</tr>
<tr>
<td>end of a queue</td>
<td>QUEUE</td>
<td>37</td>
<td></td>
<td>STACK</td>
<td>40</td>
</tr>
<tr>
<td>end of a stack</td>
<td>STACK</td>
<td>32</td>
<td>random</td>
<td>ACCESS</td>
<td>7</td>
</tr>
<tr>
<td>entity</td>
<td>NODE</td>
<td>4</td>
<td></td>
<td>ACCESS</td>
<td>15</td>
</tr>
<tr>
<td>field length</td>
<td>NODE</td>
<td>44</td>
<td>rear of a queue</td>
<td>QUEUE</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>NODE</td>
<td>68</td>
<td>record</td>
<td>QUEUE</td>
<td>9</td>
</tr>
<tr>
<td>fifo</td>
<td>QUEUE</td>
<td>37</td>
<td>top of a stack</td>
<td>STACK</td>
<td>5</td>
</tr>
<tr>
<td>first node</td>
<td>LLIST</td>
<td>25</td>
<td></td>
<td>STACK</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>LLIST</td>
<td>44</td>
<td></td>
<td>STACK</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>LLIST</td>
<td>47</td>
<td></td>
<td>STACK</td>
<td>40</td>
</tr>
<tr>
<td>following node</td>
<td>LLIST</td>
<td>25</td>
<td>word length</td>
<td>NODE</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>16</td>
<td></td>
<td>NODE</td>
<td>44</td>
</tr>
<tr>
<td>front of a queue</td>
<td>QUEUE</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>QUEUE</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>Module</td>
<td>Record</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>--------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>absolute machine address</td>
<td>PRIM1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>computer</td>
<td>PRIM1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>computer concepts</td>
<td>PRIM1</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consecutive words of memory</td>
<td>PRIM1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td>PRIM1</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>empty set</td>
<td>PRIM2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>finite set</td>
<td>PRIM2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear order</td>
<td>PRIM2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>memory location</td>
<td>PRIM1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ordered set</td>
<td>PRIM2</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set</td>
<td>PRIM2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set theory</td>
<td>PRIM2</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>word</td>
<td>PRIM1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PRIM1</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Prerequisite area.
Table 20. Words to be Translated Prior to the Decomposition Process

<table>
<thead>
<tr>
<th>Word to be Replaced</th>
<th>Resulting Word or Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>accesses</td>
<td>access</td>
</tr>
<tr>
<td>accessing</td>
<td>access</td>
</tr>
<tr>
<td>addresses</td>
<td>address</td>
</tr>
<tr>
<td>allocating</td>
<td>allocate</td>
</tr>
<tr>
<td>beads</td>
<td>bead</td>
</tr>
<tr>
<td>computers</td>
<td>computer</td>
</tr>
<tr>
<td>contents</td>
<td>content</td>
</tr>
<tr>
<td>deleting</td>
<td>delete</td>
</tr>
<tr>
<td>deletions</td>
<td>delete</td>
</tr>
<tr>
<td>ends</td>
<td>end</td>
</tr>
<tr>
<td>entities</td>
<td>entity</td>
</tr>
<tr>
<td>fields</td>
<td>field</td>
</tr>
<tr>
<td>following</td>
<td>follow</td>
</tr>
<tr>
<td>follows</td>
<td>follow</td>
</tr>
<tr>
<td>inserting</td>
<td>insert</td>
</tr>
<tr>
<td>insertions</td>
<td>insert</td>
</tr>
<tr>
<td>items</td>
<td>item</td>
</tr>
<tr>
<td>lengths</td>
<td>length</td>
</tr>
<tr>
<td>links</td>
<td>link</td>
</tr>
<tr>
<td>lists</td>
<td>list</td>
</tr>
<tr>
<td>locating</td>
<td>locate</td>
</tr>
<tr>
<td>machines</td>
<td>machine</td>
</tr>
<tr>
<td>names</td>
<td>name</td>
</tr>
<tr>
<td>nodes</td>
<td>node</td>
</tr>
<tr>
<td>operations</td>
<td>operation</td>
</tr>
<tr>
<td>ordered</td>
<td>order</td>
</tr>
<tr>
<td>orders</td>
<td>order</td>
</tr>
<tr>
<td>pointers</td>
<td>pointer</td>
</tr>
<tr>
<td>preceding</td>
<td>precede</td>
</tr>
<tr>
<td>queues</td>
<td>queue</td>
</tr>
<tr>
<td>records</td>
<td>record</td>
</tr>
<tr>
<td>stacks</td>
<td>stack</td>
</tr>
<tr>
<td>totally</td>
<td>total</td>
</tr>
<tr>
<td>within</td>
<td>in the middle</td>
</tr>
<tr>
<td>words</td>
<td>word</td>
</tr>
</tbody>
</table>
The PI Modules

One PI module was written for each definition as specified in Chapter VII. Listings of the resulting PI modules are given below. A TL or "title" operation code has been introduced to present the name of each module. In determining the record number of a response, the TL entry is not counted. Thus, for example, record 4 of the module NODE contains the DF entry beginning "WHAT WE ARE HERE CALLING."
<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DF A NODE IS ONE OR MORE CONSECUTIVE WORDS OF COMPUTER MEMORY</td>
<td>NO</td>
</tr>
<tr>
<td>DF EVERY NODE MUST BEGIN AT THE BEGINNING OF A COMPUTER WORD</td>
<td>TRUE</td>
</tr>
<tr>
<td>DF WHAT WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
<tr>
<td>DF WE ARE HERE CALLING A NODE HAS BEEN CALLED A &quot;RECORD&quot; OR &quot;FIELD&quot;</td>
<td>MAYBE</td>
</tr>
</tbody>
</table>
Look at the definition of a node. The correct answer is...

If nodes do not have names, they are referred to by their address, which is also called a link or pointer to a node.

It is important to distinguish between the name of a field and the contents of a field. The contents of a field in a node may represent numbers, alphabetic characters, links, or anything else that the programmer may desire.

Nodes are basic data elements used to construct linear lists and other data structures.
A field is a named portion of a node.
A field can be shorter than a computer word.
A field can be longer than a computer word.
A field can be 5 1/2 words long.

That's right.
That's right.
That's right.
That's right.

Yes or no?
A linear list is a set of nodes whose structural properties involve only the linear order. A linear list is a finite set of nodes. A linear list is said to be empty if it contains zero nodes. A linear list is said to be non-empty if it contains at least one node.

For a linear list, node 1 is the first node, node n is the last node, node x[i] is followed by node x[i+1], and node x[i] precedes node x[i-1]. The linear list is ordered by the linear order involved in node addresses.

You are thinking of finite sets; you are right.
笕/HISGMW 04/20/72 01:45 PM

100* TL LINK
200* LA START
300* DF THE ADDRESS OF A NODE, ALSO CALLED A LINK OR POINTER TO THAT
400* NODE, IS THE MEMORY LOCATION OF ITS FIRST WORD.
500* BP *NODE *COMPUTER WORD *MEMORY LOCATION OF A WORD.
600* DF WE WILL ASSUME THAT ALL MEMORY LOCATIONS ARE ABSOLUTE
700* MACHINE ADDRESSES; THEREFORE THE ADDRESS OF A NODE WILL
800* ALWAYS BE AN ABSOLUTE MACHINE ADDRESS.
900* BP *ABSOLUTE MACHINE ADDRESS
1000* GU IF A NODE IS IN MEMORY LOCATION 200, WHAT IS THE POINTER TO
1100* THAT NODE?
1200* CA 200
1300* UN CORRECT
1400* UN READ THE DEFINITION AND ANSWER AGAIN.
1500* UN PERHAPS YOU DID NOT UNDERSTAND THE DEFINITION. LETS REPEAT A
1600* LITTLE.*
1700* BH START
1800* GU IF A NODE CONSISTS OF WORDS 101-105 OF COMPUTER MEMORY, WHAT
1900* GU IS THE ADDRESS OF THAT NODE?
2000* CA 101
2100* TY RIGHT
2200* MA 101 = WHAT IS THE FIRST WORD OF THE NODE?
2400* UN BY THE FIRST WORD WE MEAN THE LEFT-MOST WORD OF A NODE
2500* OF CONSISTING OF WORDS 101 102 103 104 105.
2600* DF NOTE THAT THE ADDRESS OF A NODE WAS DEFINED TO BE THE MEMORY
2700* LOCATION OF ITS FIRST WORD - NOT FIELD. A NODE MUST BEGIN
2800* AT THE BEGINNING OF ITS FIRST WORD AND END AT THE END OF ITS
2900* LAST WORD. WE DO NOT DEFINE THE ADDRESS OF A FIELD BECAUSE
3000* FIELDS MAY BEGIN AND/OR END IN THE MIDDLE OF A WORD.
3100* BP *FIELD
3200* CL LINK
3300* FP THE FACT THAT A POINTER TO ONE NODE CAN BE STORED IN ANOTHER
3400* NODE PERMITS US TO REPRESENT COMPLEX DATA STRUCTURES.
3500* DF A COMPUTER BY THE USE OF WHAT IS CALLED LINKED
3600* ALLOCATION, THIS IS ONE TYPE OF STORAGE ALLOCATION WHICH IS
3700* USED FOR LINEAR LISTS.
A pointer or link is a number, and this number may be the contents of a field in another node. A field which contains a link or pointer is often called a link field or a pointer field.

If the node with address 500 has a link field named y, and y contains a link to the node with address 357, what is the content of the link field y?

Very good.

No = y is the name of the field, what is the content of the field?

What is the content of the field?

One question that sometimes bothers students is this:

How can the machine tell which field is a pointer field?

Does this question bother you?

The answer is that the machine cannot tell.

The problem is that the machine cannot tell which fields contain numbers used as pointers. It is the responsibility of the programmer to know which fields contain pointers and to use the contents of all fields correctly. For example, a field named "Jane Smith" is the field a pointer to.

Correct.

Yes.

Maybe.

You are confused.

Please answer yes or no.

Yes or no.

If a field contains "101", is the field a pointer field?

(Yes, no, or maybe) Yes.

Maybe.

No.

Not necessarily - try again.

Yes, no, or maybe.

The field might be a pointer field, but it might be anything else which happens to have as its value the number 101. It is impossible to say unless we know how the field is used.
200* OF STORAGE ALLOCATION IS THE SELECTION OF MEMORY LOCATIONS IN
300* WHICH TO STORE DATA.
400* IF IN ALLOCATING STORAGE FOR A LINEAR LIST, WE WILL ASSUME THAT
500* EACH NODE IN THE LINEAR LIST REQUIRES C CONSECUTIVE WORDS
600* OF MEMORY, WHERE C IS A CONSTANT FOR THE LINEAR LIST.
700* IF MEMOY, WHERE C IS A CONSTANT FOR THE LINEAR LIST.
800* IF TRUE OR FALSE. NOT ALL NODES IN A LINEAR LIST CONTAIN THE SAME
900* NUMBER OF WORDS.

1000* IF TRUE
1100* IF YES
1200* IF RIGHT
1300* IF FALSE
1400* IF NO
1500* IF RIGHT
1600* IF EQUAL TO TRUE
1700* IF FALSE
1800* IF R TRUE OR FALSE?
1900* IF TRUE OR FALSE?
2000* IF TRUE OR FALSE?
2100* IF TRUE OR FALSE?
2200* IF TRUE OR FALSE?
2300* IF TRUE OR FALSE?
2400* IF TRUE OR FALSE?
2500* IF TRUE OR FALSE?
2600* IF TRUE OR FALSE?
2700* IF TRUE OR FALSE?
2800* IF TRUE OR FALSE?
2900* IF TRUE OR FALSE?
3000* IF TRUE OR FALSE?
3100* IF TRUE OR FALSE?
3200* IF TRUE OR FALSE?
3300* IF TRUE OR FALSE?
3400* IF TRUE OR FALSE?
3500* IF TRUE OR FALSE?
3600* IF TRUE OR FALSE?
3700* IF TRUE OR FALSE?
3800* IF TRUE OR FALSE?
3900* IF TRUE OR FALSE?
4000* IF TRUE OR FALSE?
ALLOCATION FOR LINEAR LISTS IS A TYPE OF STORAGE IN CONSECUTIVE MEMORY LOCATIONS.

WITH SEQUENTIAL ALLOCATION, THE ADDRESS OF NODE X(K+1), FOR
K GREATER THAN 1, IS THE ADDRESS OF NODE X(K) PLUS C, WHERE
C IS THE NUMBER OF WORDS IN EACH NODE. THUS WE HAVE

WHERE LOC(X(K)) DENOTES THE ADDRESS OF NODE X(K).

IF LOC(X(K)) = 100 AND C = 1 THEN WHAT IS LOC(X(6))?

CA = 101

TY RIGHT

UN REREAD THE PREVIOUS MATERIAL AND TRY AGAIN.

WHERE LO IS A CONSTANT CALLED THE BASE ADDRESS, THE LOCATION OF AN ARTIFICIALLY ASSUMED NODE X(0)

IF LO = 101 AND C = 2 WHAT IS LOC(X(3))?

CA = 107

TY VERY GOOD

LO = 100

TY NO = LO IS 101, NOT 100.

UN HINT: LO + C * J = 101 + 2 * 3 =

TY RIGHT

QU TRUE OR FALSE: THE NODE X(0) IS STORED IN MEMORY LOCATION

LO =

CA = FALSE

CB = TRUE

CA = RIGHT

CA = TRUE

CA = YES

TY NO: X(0) IS AN ARTIFICIALLY ASSUMED NODE WHICH DOES NOT

REALLY EXIST, AND IT IS NOT STORED IN ANY LOCATION.

THE ANSWER IS...

UN TRUE OR FALSE:

QU CONSIDER A LINEAR LIST WITH 5 NODES:

IF LO = 500 AND C = 3, WHAT IS LOC(X(N))?

CA = 101

TY RIGHT

UN HINT: LO + C * J =

UN LO + C * J = 500 + 3 * 5 =

UN YOUR ARITHMETIC IS AWFUL, TRY AGAIN.
ALLOC/RISGWLW 06/20/72 01:44 PM

100* TL LALLOC
200* DF LINKED ALLOCATION FOR LINEAR LISTS IS A TYPE OF STORAGE
300* ALLOCATION IN WHICH EACH NODE IN THE LIST CONTAINS A POINTER
400* TO THE NEXT NODE IN THE LIST.
500* BP *LINKED LIST *NODE *POINTER *STORAGE ALLOCATION
600* DF A LINEAR LIST WITH LINKED ALLOCATION IS FREQUENTLY CALLED A
700* LINKED LIST.
600* QU TRUE OR FALSE? THE ADDRESS OF NODE X[K] IS ALWAYS GREATER
900* THAN THE ADDRESS OF NODE X[K] FOR ALL K LESS THAN N.
1000* BP *ADDRESS OF A NODE
1100* CA FALSE.
1200* CB NO
1300* TB RIGHT
1400* TY TRUE
1500* MA TRUE
1600* TB YES
1700* TY THINK ABOUT IT AND ANSWER AGAIN.
1900* QU TRUE OR FALSE? EVERY NODE IN A LINKED LIST MUST CONTAIN A
2000* POINTER FIELD.
2100* BP *POINTER FIELD
2200* CA TRUE
2300* CB YES
2400* TB RIGHT
2500* TY CORRECT
2600* WA FALSE
2700* WM NO
2800* TY THINK ABOUT IT AND ANSWER AGAIN.
2900* QU IF THE ADDRESS OF NODE X[K] IS 200 AND THE ADDRESS OF NODE
3000* X[K+1] IS 127 WHAT IS THE CONTENT OF THE POINTER FIELD OF
3100* NODE X[K+1]?
3200* BP *POINTER FIELD
3300* CA 127
3400* CB NO
3500* TB RIGHT
3600* TY TRUE
3700* WC 200
3800* UN READ THE MATERIAL AND TRY AGAIN.
3900* QU LINKED ALLOCATION TAKES UP ADDITIONAL MEMORY SPACE FOR
4000* LINKS. HOWEVER, WE FREQUENTLY HAVE A DUMMY FIELD WHICH CAN
4100* BE USED AS THE LINK FIELD.
4200* *DUMMY FIELD
OPERNS/RISGWLM 04/20/72 01144 PM

100 TL OPERNS
200 DF THERE ARE MANY POSSIBLE OPERATIONS ON LINEAR LISTS. BASIC
300 OPERATIONS MIGHT INCLUDE:
400 1. ACCESS THE KTH NODE TO EXAMINE AND/OR CHANGE THE
500 CONTENTS OF ITS FIELDS,
600 2. INSERT A NEW NODE JUST BEFORE (OR AFTER) THE KTH NODE,
700 3. DELETE THE KTH NODE,
800 BP *LINEAR LIST *NODE *FIELD
900 CL ACCESS
1000 CL INSERT
1100 CL DELETE
1200 FP FREQUENTLY, INSERTIONS, DELETIONS OCCUR ONLY
1300 AT THE ENDS OF A LINEAR LIST. IN FACT, LINEAR LISTS IN
1400 WHICH INSERTIONS AND DELETIONS OCCUR ONLY AT THE ENDS ARE
1500 ENCOUNTERED SO FREQUENTLY THEY HAVE BEEN GIVEN SPECIAL
1600 NAMES. TWO SUCH LISTS ARE STACKS AND QUEUES.

ACCESS/RISGWLM 04/20/72 01145 PM

100 TL ACCESS
200 DF ACCESSING A NODE IS AN OPERATION IN WHICH A SPECIFIED NODE
300 IS LOCATED AND ITS CONTENTS EXAMINED AND/OR CHANGED.
400 BP *NODE *OPERATION
500 DF IN ORDER TO LOCATE A NODE IN A LINEAR LIST, IT IS NECESSARY
600 TO DETERMINE THE ADDRESS OF THE NODE.
700 BP *ADDRESS OF A NODE *LINEAR LIST
800 DF WITH SEQUENTIAL ALLOCATION, IT IS EASY TO ACCESS ANY NODE AT
900 RANDOM IN A LINEAR LIST BY COMPUTING ITS ADDRESS.
1000 BP *SEQUENTIAL ALLOCATION
1100 DF IN ORDER TO ACCESS THE KTH NODE (X(k)) IN A LINEAR LIST WITH LINKED
1200 ALLOCATION, IT IS NECESSARY TO ACCESS NODE X(1) TO GET THE
1300 LINK TO NODE X(2), THEN ACCESS NODE X(2) TO GET THE LINK TO
1400 NODE X(3), AND SO FORTH UNTIL X(k) IS LOCATED.
1500 BP *LINKED ALLOCATION
1600 DF IT IS FASTER TO ACCESS NODES AT RANDOM WITHIN A LINEAR LIST
1700 WHEN SEQUENTIAL ALLOCATION IS USED RATHER THAN LINKED
1800 ALLOCATION.

INSERT/RISGWLM 04/20/72 01144 PM

100 TL INSERT
200 DF INSERTING IS AN OPERATION IN WHICH A NODE IS INSERTED INTO A
300 DATA STRUCTURE SUCH AS A LINEAR LIST.
400 BP *NODE *LINEAR LIST *OPERATION
500 DF INSERTIONS FREQUENTLY OCCUR AT THE ENDS OF A LINEAR LIST.
600 BUT NODES MAY BE INSERTED ANYWHERE IN THE LIST. THE WAY IN
700 WHICH A NODE IS INSERTED IN A LIST IS HIGHLY DEPENDENT UPON
800 THE STORAGE ALLOCATION.
900 BP *ENDS OF A LINEAR LIST *STORAGE ALLOCATION
1000 DF INSERTING A NODE AT THE END OF A SEQUENTIALLY ALLOCATED LIST
1100 INVOLVES PLACING THE DATA IN THE APPROPRIATE NODE AND
1200 OBTAINING A COUNT OF THE NUMBER OF NODES IN THE LIST OR A
1300 POINT TO THE END OF THE LIST, BUT INSERTING A NODE IN A
1400 LIST THAT ALSO INVOLVES MOVING ALL SUBSEQUENT NODES IN THE
1500 LIST DOWN TO OTHER MEMORY LOCATIONS TO MAKE ROOM FOR THE
1600 NODE BEING INSERTED.
1700 BP *RANDOM LOCATION *ENDS OF A LISTS *SEQUENTIAL ALLOCATION
1800 DF INSERTING A NODE AFTER THE KTH NODE IN A LINKED LIST IS
1900 ACCOMPLISHED BY PLACING THE PREVIOUS CONTENTS OF THE POINT
2000 FIELD OF THE KTH NODE IN THE POINTER FIELD OF THE NEW NODE,
2100 AND THEN PLACING THE ADDRESS OF THE NEW NODE IN THE POINTER
2200 FIELD OF THE KNO X(k)
2300 BP *ADDRESS OF A NODE *POINTER FIELD *LINKED LIST
2400 DF IT IS EASY TO INSERT A NODE IN THE MIDDLE OF A LINKED LIST.
2500 THE SAME OPERATION WOULD BE VERY TIME CONSUMING IN A LONG
2600 LINEAR LIST WITH SEQUENTIAL ALLOCATION.
DELETE/RIGWLM 04/20/72 01144 PM

100* TL DELETE
200* DF DELETING A NODE IS AN OPERATION IN WHICH A NODE IS REMOVED
300* BP FROM A DATA STRUCTURE SUCH AS A LINEAR LIST.
400* BP *NODE *LINEAR LIST *OPERATION
500* DF ALTHOUGH DELETIONS FREQUENTLY OCCUR AT THE ENDS OF A LINEAR
600* LIST, IT IS SOMETIMES NECESSARY TO DELETE A NODE WITHIN THE
700* LIST, THE WAY IN WHICH THE DELETION IS ACCOMPLISHED DEPENDS
800* UPON THE STORAGE ALLOCATION USED FOR THE LIST.
900* BP *ENDS OF A LINEAR LIST *STORAGE ALLOCATION
1000* DF DELETING A NODE FROM THE END OF A LINEAR LIST WITH
1100* SEQUENTIAL ALLOCATION ONLY INVOLVES CHANGING THE COUNT OF
1200* NODES WITHIN THE LIST OR A POINTER TO THE END OF THE LIST.
1300* HOWEVER, DELETING A NODE WITHIN THE LIST INVOLVES MOVING THE
1400* CONTENTS OF ALL SUBSEQUENT NODES UP TO DIFFERENT LOCATIONS
1500* TO ELIMINATE THE "HOLE" LEFT BY THE DELETED NODE.
1600* BP *SEQUENTIAL ALLOCATION
1700* DF TO DELETE NODE X(K) FROM A LINKED LIST, IT IS NECESSARY
1800* TO ACCESS NODE X(K-1) AND CHANGE THE POINTER FIELD SO THAT
1900* NODE X(K-1) NOW POINTS TO THE NODE WHICH PREVIOUSLY FOLLOWED
2000* NODE X(K).
2100* BP *POINTER FIELD *LINKED LIST
2200* DF IT IS EASY TO DELETE AN ITEM FROM WITHIN A LINKED LIST, BUT
2300* WITH SEQUENTIAL ALLOCATION SUCH A DELETION FREQUENTLY
2400* INVOLVES MOVING A LARGE PART OF THE LIST UP TO DIFFERENT
2500* LOCATIONS,
STACK/RISGWLM 04/20/72 01145 PM

213

STACK/RISGWLM 04/20/72 01146 PM

100 TL STACK
200 OF A STACK IS A LINEAR LIST IN WHICH ALL INSERTIONS AND
300 DELETIONS (AND USUALLY ALL ACCESSES) ARE MADE AT THE SAME
400 END OF THE LIST.
500 BP *LINEAR LIST *ENDS OF A LINEAR LIST
600 BP *ACCESS INSERTIONS AND DELETIONS
700 CL QUEUE
1000 OF THE DIFFERENCE BETWEEN A STACK AND A QUEUE IS THAT THE NEXT
1100 ITEM TO BE REMOVED FROM A STACK IS THE LAST ITEM PLACED ON
1200 THE STACK, WHILE THE NEXT ITEM TO BE REMOVED FROM A QUEUE IS
1300 THE ITEM WHICH HAS BEEN THERE THE LONGEST.
1400 *ITEM

STACK/RISGWLM 04/20/72 01146 PM

100 TL STACK
200 OF A STACK IS A LINEAR LIST FOR WHICH ALL INSERTIONS AND
300 DELETIONS (AND USUALLY ALL ACCESSES) ARE MADE AT THE SAME
400 END OF THE LIST.
500 BP *LINEAR LIST *ENDS OF A LINEAR LIST
600 BP *ACCESS INSERTIONS AND DELETIONS
700 CL STACK
800 TL STACK
900 DF A STACK MAY BE THOUGHT OF AS BEING LIKE A STACK OF TRAYS IN
1000 A CAFETERIA, WHEN WE REMOVE A TRAY FROM THE STACK WE REMOVE
1100 THE TOP TRAY, WHEN WE PLACE A TRAY ON THE STACK WE PLACE
1200 IT ON THE TOP OF THE STACK.
1300 DF WHEN TALKING ABOUT STACKS IN A COMPUTER WE REFER TO THE ENDS
1500 NODE MOST RECENTLY PLACED ON THE STACK IS SAID TO BE ON THE
1600 TOP.
1700 EP *NODE
1800 DU AT WHICH END OF A STACK DO INSERTIONS AND DELETIONS OCCUR>
1900 CA TOP
2000 CA BOTTOM
1900 TY NO, TRY AGAIN,
2000 IN THE TOP, BOTTOM>
2100 DF MOST ACCESS OCCUR AT THE TOP OF A STACK, BUT
2200 IT IS POSSIBLE TO ACCESS A NODE WITHIN A STACK.
2300 BU IF THE LAST ITEM PLACED ON THE STACK
2400 X[1], X[2], X[3], X[4], X[5]
2500 WAS X[5] AND WE DELETE A NODE, WHICH NODE WILL BE DELETED>
2600 BU X[5]
2700 BU X[5]
2800 BU X[5]
2900 BU X[5]
3000 BU X[5]
3100 BU X[5]
3200 UN X[1], X[2], X[3]
3300 DF ALL INSERTIONS AND DELETIONS OCCUR AT THE SAME END
3400 OF A STACK, THE ITEM REMOVED FROM THE STACK IS ALWAYS THE
3500 ITEM MOST RECENTLY PLACED ON THE STACK, FOR THIS REASON
3600 STACKS ARE SOMETIMES CALLED LAST-IN-FIRST-OUT OR LIFO LISTS.
3700 DF STACKS HAVE ALSO BEEN CALLED PUSH-DOWN LISTS, REVERSION
3800 STORAGE CELLS, NESTING STORES, PILES, AND EVEN YO-YO
3900 LISTS.
4000 DF PEOPLE FREQUENTLY SAY THAT THEY "POP" A STACK MEANING THAT
4100 THEY DELETE THE TOP ITEM, OR THAT THEY "PUSH DOWN" A STACK
4200 MEANING THAT THEY PLACE AN ITEM ON THE STACK.
A queue is a linear list for which all insertions are made at one end of the list and all deletions (and usually all accesses) are made at the other.

A queue may be thought of as being like a line of people. People join the rear of the queue, called the rear, and leave the front, called the front. The node most recently placed in the queue is at the rear of the queue. Computer queues are also called first-in-first-out or FIFO lists, because the item removed from a queue is always the "oldest".
100* TL PRIM1
200* OF COMPUTER CONCEPTS ARE PREREQUISITE TO THIS UNIT.
300* IF YOU ARE NOT FAMILIAR WITH ADDRESSING IN
400* PRIMARY (CORE) MEMORY.
500* A MEMORY LOCATION IS THE ABSOLUTE MACHINE ADDRESS OF A
600* COMPUTER WORD. SEE KNUTH CHAPTER 1 IF THIS IS NOT CLEAR.
700* CONSECUTIVE WORDS OF MEMORY ARE WORDS WITH CONSECUTIVE
800* ADDRESSES, SEE KNUTH CHAPTER 1 FOR FURTHER INFORMATION
900* ON MEMORY ADDRESSES.
1000* SEE KNUTH FOR A DEFINITION OF "DATA".

100* TL PRIM2
200* OF A FEW SET THEORETIC CONCEPTS ARE PREREQUISITE TO
300* THIS UNIT, SEE STOLL CHAPTER 1.
400* AN EMPTY SET IS A SET WHICH CONTAINS NO ELEMENTS.
500* SEE STOLL PAGES 10-11.
600* IF YOU DO NOT KNOW WHAT A FINITE SET IS, PLEASE STUDY
700* STOLL CHAPTER 1.
800* A LINEAR ORDER IS ALSO CALLED A SIMPLE ORDER.
900* SEE STOLL PAGE 50.
1000* A TOTALLY ORDERED SET IS A SET WITH A LINEAR OR SIMPLE
1100* ORDER DEFINED ON IT. SEE STOLL CHAPTER 1.
The Script

The script used in the experimental runs with the final system was produced from the vocabulary and the PI modules as prescribed in Chapter VII. In the following listing of the script, a "$" follows each script element as required by the GTL LISP system.
(BEGIN)

100* (NODE/MEM)
200* (FIELD/MEM)
300* (ADDRESS/MEM)
400* (LINK/MEM)
500* (POINTER/MEM)
600* (LINEAR/MEM)
700* (STORAGE/MEM)
800* (ALLOCATION/MEM)
900* (LINKED/MEM)
1000* (SEQUENTIAL/MEM)
1100* (OPERATIONS/MEM)
1200* (ACCESS/MEM)
1300* (STACK/MEM)
1400* (DEFINITE/MEM)
1500* (TARGET/MEM)
1600* (FUNCTION/MEM)
1700* (FUNCTION/LET)
1800* (FUNCTION/LET)
1900* (FUNCTION/LET)
2000* (FUNCTION/LET)
2100* (FUNCTION/LET)
2200* (FUNCTION/LET)
2300* (FUNCTION/LET)
2400* (FUNCTION/LET)
2500* (FUNCTION/LET)
2600* (FUNCTION/LET)
2700* (FUNCTION/LET)
2800* (FUNCTION/LET)
2900* (FUNCTION/LET)
3000* (FUNCTION/LET)
3100* (FUNCTION/LET)
3200* (FUNCTION/LET)
3300* (FUNCTION/LET)
3400* (FUNCTION/LET)
3500* (FUNCTION/LET)
3600* (FUNCTION/LET)
3700* (FUNCTION/LET)
3800* (FUNCTION/LET)
3900* (FUNCTION/LET)
4000* (FUNCTION/LET)
4100* (FUNCTION/LET)
4200* (FUNCTION/LET)
4300* (FUNCTION/LET)
4400* (FUNCTION/LET)
4500* (FUNCTION/LET)
4600* (FUNCTION/LET)
4700* (FUNCTION/LET)
4800* (FUNCTION/LET)
4900* (FUNCTION/LET)
5000* (FUNCTION/LET)
5100* (FUNCTION/LET)
5200* (FUNCTION/LET)
5300* (FUNCTION/LET)
5400* (FUNCTION/LET)
5500* (FUNCTION/LET)
5600* (FUNCTION/LET)
5700* (FUNCTION/LET)
5800* (FUNCTION/LET)
5900* (FUNCTION/LET)
6000* (FUNCTION/LET)
6100* (FUNCTION/LET)
6200* (FUNCTION/LET)
6300* (FUNCTION/LET)
6400* (FUNCTION/LET)
6500* (FUNCTION/LET)
6600* (FUNCTION/LET)
6700* (FUNCTION/LET)
6800* (FUNCTION/LET)
6900* (FUNCTION/LET)
7000* (FUNCTION/LET)
7100* (FUNCTION/LET)
7200* (FUNCTION/LET)
7300* (FUNCTION/LET)
7400* (FUNCTION/LET)
7500* (FUNCTION/LET)
7600* (FUNCTION/LET)
7700* (FUNCTION/LET)
7800* (FUNCTION/LET)
7900* (FUNCTION/LET)
8000* (FUNCTION/LET)
8100* (FUNCTION/LET)

(END)
APPENDIX B

THE COMPLETE RECORD OF A STUDENT INTERACTION WITH GITIT

In this appendix we give the complete record of a student interaction with the system in the course of a six-day experiment. Before presenting the record of the dialogue between the student and the computer, we will briefly describe the experiment and the instructions given the student who served as a subject in the experiment.

On Day 1 the subject was given the following instructions:

Give the best definition you can of each of these terms. Then rate your definition using A, B, C, D, F.

Do not look ahead and do not go back to change either a definition or a rating.

When you have completed all definitions you will be given a copy of your definitions. Between now and the session tomorrow, try to improve your definitions as much as you can.

Do not discuss these definitions with any one or use any books or other reference materials.

He was given the following terms to define:

stack
bead
allocation
pop
node
lifo
push down
access
empty node

The terms were each written on a separate sheet of paper, and the student went through the terms in the above sequence. He was permitted
to take as long as he needed on each definition.

On the next day, Day 2, the subject was given a new set of terms to define, again proceeding through the terms one at a time. The second set of terms was:

- field
- fifo
- record
- linear list
- address of a field
- dummy field
- link
- ends of a list

When the subject had completed the second set of definitions, he was asked to go back over all terms in both sets changing any definitions and/or grades he wished until he was satisfied that he had improved all definitions as much as he could.

The subject's task on the machine was to improve his definitions. He was encouraged to use a paraphrase rather than the definition obtained from the system, and assured that his definition was to be considered correct if it defined the same concept in different terms. It was stressed that he should understand any change he made and that he should not accept a different definition from the system unless he was sure that he understood the new definition.

The next step in the experiment used the CAI system GITIT. The student was given an explanation of the discussion and tutorial modes, but was told nothing about the instructional materials. He was then permitted to use the system, using the two modes in any way he wished. The experimenter keyed in the student questions and responses.

Whenever the subject wished to change a definition or grade, the
system was stopped to permit him to do so. These stops were numbered within each run, and the stop number was written both on the computer printout and by any changes the student made at that stop.

The run continued until the subject was satisfied with all definitions in the first two sets. The resulting run, Run 1, is given in Figure 33 in this appendix. This concluded the first two days of experimentation.

On Day 3, approximately a week later, the subject was given a third list of terms as follows:

- delete
- linked list
- sequential allocation
- name of a field
- empty list
- link field
- contents of a node
- insert
- linked allocation
- queue
- name of a node

He was again asked to go through the terms once writing definitions, and then permitted to go back over all terms as needed.

Then he returned to the system for Run 2 (see Figure 34). His task was to refine all definitions in preparation for a test over the material. When he was satisfied with all of his definitions, he was given the following instructions:

Please review everything very carefully paying special attention to the grade you have assigned. You may change your grades and/or your definitions if you wish to do so, and you may use the machine again if you wish.

When you are satisfied with your definitions, you will be given some questions to answer. You may use your definitions in answering the questions, but you must not use any other materials.
He decided that his definitions were adequate. He was then given the following instructions:

What are the similarities and differences between the following paired concepts?

In determining your answer, you may use your definitions, but please do not use any other references.

Grade each of your answers, A, B, C, D, F. If you wish, you may have access to the computer tomorrow to review or ask any further questions you may have.

The paired concepts were as follows:

- node vs. field
- linked allocation vs. sequential allocation
- accessing vs. inserting
- accessing vs. deleting
- stack vs. queue
- name of a field vs. content of a field
- name of a field vs. address of a node
- linked list vs. linear list

The subject returned on Day 4 with his answers and a request to use the system again. The resulting run, Run 3, is shown in Figure 35 below.

After the results had been analyzed, the subject was asked to return (Day 5) for a few questions. He was asked to give examples of the following concepts:

- linear list
- linked allocation
- stack
- queue

Finally, the subjects were each recalled after a period of several weeks. At this time (Day 6) the subject was given the following instructions:

Write definitions of each of the following terms. Give the definition you would use if you were teaching the term to
someone else, that is, a definition which you are comfortable with and could explain.

He was then asked to define:

```
queue
node
dummy field
linear list
linked list
```

After he had written the definitions, he was asked to define orally all terms which he used in the definition of the term "queue."

He was then given the following instructions:

Assume that you are directing a student to use the system and only the tutorial mode is available. In which order would you suggest that he use the following modules?

Indicate the order by numbering the following module names. You may use the same number in more than one module in a "don't care" situation.

The subject was given the following list of modules:

```
ALLOC    storage allocation
LLIST    linear list
NODE     node
STACK    stack
LINK     link, pointer or address of a node
OPERNS   operations
QUEUE    queue
```

The subject was then asked, orally, whether or not a queue as defined in the system could exist outside of a computer. The results of the case study are discussed in Chapter V.

Figures 33, 34, and 35 give the complete protocol for one subject for Runs 1, 2, and 3, respectively. All level 1 computer processes are indicated in these figures to show the embedding structure. Level 0 computer processes are also indicated on the figures for Run 1. Stops are shown on the protocol as encircled numbers. The definitions
recorded by this subject during the first four days of experimentation are given in Table 21. The stop numbers shown in these tables correspond to the stops indicated on the protocol for the corresponding run.
RUN GITIT-
RUNNING

YOU ARE USING GITIT, AN EXPERIMENTAL CAI SYSTEM WHICH CAN
OPERATE IN EITHER A DISCUSSION (STUDENT-CONTROLLED) MODE,
OR IN A TUTORIAL (MACHINE-CONTROLLED) MODE. YOU MAY CAUSE
GITIT TO SHIFT MODES AT ANY TIME BY USING THE APPROPRIATE
COMMANDS.

ONE MOMENT PLEASE... WE WILL START IN DISCUSSION MODE. TYPE *HELP FOR ASSISTANCE.
WHAT SHALL WE TALK ABOUT?

DISTACKS-
A STACK IS A LINEAR LIST FOR WHICH ALL INSERTIONS AND
DELETIONS (AND USUALLY ALL ACCESSES) ARE MADE AT THE SAME
END OF THE LIST. [STACK ]

LINEAR LISTS-
A LINEAR LIST IS A SET OF N GREATER THAN OR EQUAL TO ZERO
NODES
X[1], X[2], ..., X[N]
WHOSE STRUCTURAL PROPERTIES INVOLVE ONLY THE LINEAR ORDER
OF THE NODES. [LIST ]

WHERE A NODE CONSIST OF A SINGLE WORD?

NAY A NODE CONSIST OF A SINGLE WORD?

Figure 33. The Complete Protocol for S_1 on Run 1
Is it possible to have an empty node, that is, a node consisting of zero words?

Right

A node must consist of at least one word, and must be divided into (at least one) field. The fields may contain zeroes or garbage, but the node cannot be empty.

RETURN

LEAVING MODULE NODE

Fields

A field is a named portion of a node. (Field)

Parent Field

Entering Module Field

A field is a named portion of a node.

Node

Can a field be shorter than a computer word?

Yes

That's right

Can a field be longer than a computer word?

Yes

Correct

Can a field be 5 1/2 words long?

RETURN

LEAVING MODULE FIELD

Figure 33. (Continued)
A system failure occurred at this point. The necessary control commands and responses were given to reset the program to the previous state.

Figure 33. (Continued)
WHICH IS THE MOST PRECISE DEFINITION OF A LINEAR LIST?

A LINEAR LIST IS...
1. A FINITE SET OF NODES.
2. A TOTALLY ORDERED SET OF NODES.
3. A (POSSIBLY EMPTY) FINITE SET OF NODES WHICH IS TOTALLY ORDERED.
•FINITE SET •TOTALLY ORDERED SET •EMPTY SET

CORRECT

NUMBER 3 IS THE BEST CHOICE BECAUSE IT SPECIFIES THAT THE SET MUST BE FINITE AND TOTALLY ORDERED, AND THAT IT MAY BE EMPTY. NUMBER 2 IS ALMOST CORRECT, BUT FAILS TO SPECIFY THAT THE SET IS FINITE.

FOR N GREATER THAN ZERO, X(1) IS THE FIRST NODE.
FOR K GREATER THAN 1, X(K) IS PRECEDED BY X(K-1).
FOR K LESS THAN N, X(K) IS FOLLOWED BY X(K+1).
X(N+1) IS THE LAST NODE.

CAN A LINEAR LIST BE EMPTY, THAT IS CAN A LINEAR LIST CONTAIN ZERO NODES?

YES

Another system failure occurred here. The student elected not to reinitiate the module LLIST because he was ready to return to the discussion mode.

Figure 33. (Continued)
A stack is a linear list for which all insertions and deletions (and usually all accesses) are made at the same end of the list. [STACK]

When talking about stacks in a computer we refer to the ends of the list as the top and the bottom of the stack. The node most recently placed on the stack is said to be on the top. [STACK]

A stack may be thought of as being like a stack of trays in a cafeteria. When we remove a tray from the stack we remove the top tray. When we place a tray on the stack we place it on the top of the stack. [STACK]

When talking about stacks in a computer we refer to the ends of the list as the top and the bottom of the stack. The node most recently placed on the stack is said to be on the top. [NODE]

At which end of a stack do insertions and deletions occur?

What shall we talk about?

I thought you said insertions and deletions could occur at either end of the stack. Insertions frequently occur at the ends of a linear list, but nodes may be inserted anywhere in the list. The way in which a node is inserted in a list is highly dependent upon the storage allocation. [INSERT]

Figure 33. (Continued)
A STACK IS A LINEAR LIST FOR WHICH ALL INSERTIONS AND
DELETIONS (AND USUALLY ALL ACCESSES) ARE MADE AT THE SAME
END OF THE LIST.

A STACK MAY BE THOUGHT OF AS BEING LIKE A STACK OF TRAYS IN
A CAFETERIA. WHEN WE REMOVE A TRAY FROM THE STACK WE REMOVE
THE TOP TRAY. WHEN WE PLACE A TRAY ON THE STACK WE PLACE
IT ON THE TOP OF THE STACK.

WHEN TALKING ABOUT STACKS IN A COMPUTER WE REFER TO THE ENDS
NODE MOST RECENTLY PLACED ON THE STACK IS SAID TO BE ON THE
TOP.

AT WHICH END OF A STACK DO INSERTIONS AND DELETIONS OCCUR?

RIGHT

MOST ACCESSES OCCUR AT THE TOP OF A STACK, BUT
IT IS POSSIBLE TO ACCESS A NODE WITHIN A STACK.

IF THE LAST ITEM PLACED ON THE STACK
X(1), X(2), X(3), X(4), X(5)
WAS X(5) AND WE DELETE A NODE, WHICH NODE WILL BE DELETED?

X(5)

Figure 33. (Continued)
BECAUSE ALL INSERTIONS AND DELETIONS OCCUR AT THE SAME END OF A STACK, THE ITEM REMOVED FROM THE STACK IS ALWAYS THE ITEM MOST RECENTLY PLACED ON THE STACK. FOR THIS REASON STACKS ARE SOMETIMES CALLED LAST-IN-FIRST-OUT OR LIFO LISTS.

STACKS HAVE ALSO BEEN CALLED PUSH-DOWN LISTS, REVERSION STORAGES, CELLARS, NESTING STORES, PIELES, AND EVEN YO-YO LISTS.

PEOPLE FREQUENTLY SAY THAT THEY "POP" A STACK MEANING THAT THEY DELETE THE TOP ITEM, OR THAT THEY "PUSH DOWN" A STACK MEANING THAT THEY PLACE AN ITEM ON THE STACK.

WHAT WE ARE HERE CALLING A NODE HAS BEEN CALLED A "RECORD" "ENTITY" OR "BEAD" BY SOME AUTHORS. WE WILL OCCASIONALLY SAY "ITEM" INSTEAD OF NODE. (NODE)
Can a field be shorter than a computer word?

Yes—That’s right.

Can a field be longer than a computer word?

Yes—Correct.

Can a field be 5 1/2 words long?

Yes—Sure.

Do all fields have names?

Yes—you are right.

Leaving module field

-------------

What is a dummy field?

Every node must begin at the beginning of a computer word and end at the end of a computer word. If the fields needed for a node require a total of, say, 5 1/2 words of computer memory, then we must either adjust the length of the existing fields or add a "dummy" field so that the node will only contain whole words. [Node]

Define address of a field—Note that the address of a node was defined to be the memory location of its first word—not field. A node must begin at the beginning of its first word and end at the end of its last word. We do not define the address of a field because fields may begin and/or end in the middle of a word. [Link]

Define FIFO—Queues are sometimes called first-in-first-out or FIFO lists because the item removed from a queue is always the "oldest" item. [Queue]

Figure 33. (Continued)
DEFINING ACCESS:
Accessing a node is an operation in which a specified node is located and its contents examined and/or changed. (Access)

TUTORIAL ACCESS:

ENTERING MODULE ACCESS

ACCCSSING A NODE IS AN OPERATION IN WHICH A SPECIFIED NODE IS LOCATED AND ITS CONTENTS EXAMINED AND/OR CHANGED.

NODE OPERATIONS

IN ORDER TO LOCATE A NODE IN A LINEAR LIST, IT IS NECESSARY TO DETERMINE THE ADDRESS OF THE NODE.

ADDRESS OF A NODE
LINEAR LIST

WITH SEQUENTIAL ALLOCATION IT IS EASY TO ACCESS ANY NODE AT RANDOM IN A LINEAR LIST BY COMPUTING ITS ADDRESS.

SEQUENTIAL ALLOCATION

IN ORDER TO ACCESS NODE X(i) IN A LINEAR LIST WITH LINKED ALLOCATION, IT IS NECESSARY TO ACCESS NODE X(i-1) TO GET THE LINK TO NODE X(i), THEN ACCESS NODE X(i) TO GET THE LINK TO NODE X(i+1) AND SO FORTH UNTIL WE ARRIVE AT NODE X(k).

LINKED ALLOCATION

DISCUSS:
WHAT SHALL WE TALK ABOUT?

CONTROL:
DEFINITION ALLOCATION:
LET ME TUTOR YOU ON THAT

STORAGE ALLOCATION IS THE SELECTION OF MEMORY LOCATIONS IN WHICH TO STORE DATA.

MEMORY LOCATION
DATA

IN ALLOCATING STORAGE FOR A LINEAR LIST, WE WILL ASSUME THAT EACH NODE IN THE LINEAR LIST REQUIRE C CONSECUTIVE WORDS OF MEMORY, WHERE C IS A CONSTANT FOR THE LINEAR LIST.

LINEAR LIST
NODE
CONSECUTIVE WORDS OF MEMORY

Figure 33. (Continued)
TRUE OR FALSE: ALL NODES IN A LINEAR LIST CONTAIN THE SAME NUMBER OF WORDS.

TRUE

RIGHT

ASSUME THAT YOU ARE ALLOCATING STORAGE FOR A LINEAR LIST IN WHICH EACH NODE CONSISTS OF 3 COMPUTER WORDS. WHAT IS THE VALUE OF C FOR THAT LINEAR LIST?

TRUE

RIGHT

IF C = 2, HOW MANY WORDS OF COMPUTER MEMORY ARE REQUIRED TO CONTAIN A LINEAR LIST WITH AT MOST 25 NODES?

TRUE

CORRECT

THE TWO BASIC TYPES OF STORAGE ALLOCATION ARE SEQUENTIAL ALLOCATION AND LINKED ALLOCATION.

WITH SEQUENTIAL ALLOCATION, THE ADDRESS OF NODE X(K+1), FOR K GREATER THAN 1, IS THE ADDRESS OF NODE X(K) PLUS C, WHERE C IS THE NUMBER OF WORDS IN EACH NODE. THUS WE HAVE LOC(X(K+1)) = LOC(X(K)) + C
WHERE LOC(X(K)) DENOTES THE ADDRESS OF NODE X(K).

*NODE = ADDRESS OF A NODE

Figure 33. (Continued)
LINKED ALLOCATION FOR LINEAR LISTS IS A TYPE OF STORAGE ALLOCATION IN WHICH EACH NODE IN THE LIST CONTAINS A POINTER TO THE NEXT NODE IN THE LIST.

- LINEAR LIST
- NODE
- POINTER
- STORAGE ALLOCATION

A LINEAR LIST WITH LINKED ALLOCATION IS FREQUENTLY CALLED A LINKED LIST.

TRUE OR FALSE: THE ADDRESS OF NODE X(N) IS ALWAYS GREATER THAN THE ADDRESS OF NODE X(K) FOR ALL K LESS THAN N.

ADDRESS OF A NODE

TRUE OR FALSE: EVERY NODE IN A LINKED LIST MUST CONTAIN A POINTER FIELD.

CORRECT

IF THE ADDRESS OF NODE X(K) IS 200 AND THE ADDRESS OF NODE X(K+1) IS 127 WHAT IS THE CONTENT OF THE POINTER FIELD OF NODE X(K+1)?

RIGHT

LINKED ALLOCATION TAKES UP ADDITIONAL MEMORY SPACE FOR LINKS, HOWEVER WE FREQUENTLY HAVE A DUMMY FIELD WHICH CAN BE USED AS THE LINK FIELD.

Figure 33. (Continued)
DISCUSS:
WHAT SHALL WE TALK ABOUT?
DEFINE LINK:
LET ME TUTOR YOU ON THAT

THE ADDRESS OF A NODE, ALSO CALLED A LINK OR POINTER TO THAT
NODE, IS THE MEMORY LOCATION OF ITS FIRST WORD.

ADDRESS = COMPUTER WORD = MEMORY LOCATION OF A WORD

WHERE:
TUTORIAL MODE, MODULE LINK
RETURN:
WHAT WOULD YOU LIKE TO DISCUSS NEXT?

RECORD:
WHAT WE ARE HERE CALLING A NODE HAS BEEN CALLED A "RECORD"
"ENTITY" OR "BEAD" BY SOME AUTHORS. WE WILL OCCASIONALLY
SAY "ITEM" INSTEAD OF NODE. [NODE ]

STOP:
GOOD BYE

Figure 33. (Continued)
RUN GITIT-
  RUNNING

P1

NICE TO SEE YOU AGAIN, PIVXW.
WE WILL START IN DISCUSSION MODE. TYPE *HELP FOR ASSISTANCE.
WHAT SHALL WE TALK ABOUT?
DI: LINKED LISTS-
A LINEAR LIST WITH LINKED ALLOCATION IS FREQUENTLY CALLED A
LINKED LIST. [LALLOC ]

P2

DI: CONTROL-
DI: DEFINE LINKED ALLOCATION-
LET ME TUTOR YOU ON THAT

LINKED ALLOCATION FOR LINEAR LISTS IS A TYPE OF STORAGE
ALLOCATION IN WHICH EACH NODE IN THE LIST CONTAINS A POINTER
TO THE NEXT NODE IN THE LIST.
*LINEAR LIST *NODE *POINTER *STORAGE ALLOCATION

T1=

A LINEAR LIST WITH LINKED ALLOCATION IS FREQUENTLY CALLED A
LINKED LIST.

T1=

TRUE OR FALSE: THE ADDRESS OF NODE X(N) IS ALWAYS GREATER
THAN THE ADDRESS OF NODE X(K) FOR ALL K LESS THAN N.
*ADDRESS OF A NODE

T1: FALSE-
RIGHT

TRUE OR FALSE: EVERY NODE IN A LINKED LIST MUST CONTAIN A
POINTER FIELD.
*POINTER FIELD

T1: TRUE-
CORRECT

IF THE ADDRESS OF NODE X(K) IS 200 AND THE ADDRESS OF NODE
X(K+1) IS 127 WHAT IS THE CONTENT OF THE POINTER FIELD OF
NODE X(K)?

T1: 127-
RIGHT

Figure 34. The Complete Protocol for S1 on Run 2
LINKED ALLOCATION TAKES UP ADDITIONAL MEMORY SPACE FOR LINKS. HOWEVER, WE FREQUENTLY HAVE A DUMMY FIELD WHICH CAN BE USED AS THE LINK FIELD.

WHAT WOULD YOU LIKE TO DISCUSS NEXT?

DEFINE THE NAME OF A FIELD-
IT IS IMPORTANT TO DISTINGUISH BETWEEN THE NAME OF A FIELD AND THE CONTENTS OF A FIELD. THE CONTENTS OF A FIELD IN A NODE MAY REPRESENT NUMBERS, ALPHANUMERIC CHARACTERS, LINKS, OR ANYTHING ELSE THAT THE PROGRAMMER MAY DESIRE. (NODE)

ENTERING MODULE NODE

A NODE IS ONE OR MORE CONSECUTIVE WORDS OF COMPUTER MEMORY DIVIDED INTO ONE OR MORE NAMED PARTS CALLED FIELDS.

WHAT WE ARE HERE CALLING A NODE HAS BEEN CALLED A "RECORD" "ENTITY" OR "BEAD" BY SOME AUTHORS. WE WILL OCCASIONALLY SAY "ITEM" INSTEAD OF NODE.

MAY A NODE CONSIST OF A SINGLE WORD?

YES-CORRECT

IS IT POSSIBLE TO HAVE AN EMPTY NODE, THAT IS, A NODE CONSISTING OF ZERO WORDS?

RIGHT

Figure 34. (Continued)
A node must consist of at least one word, and must be divided into (at least one) field. The fields may contain zeroes or garbage, but the node cannot be empty.

**T:**
May a node consist of the words 101 and 205?

**T:**
Yes—look at the definition carefully.

A system failure occurred at this point. The necessary control commands and responses were given to reset the program to the previous state.

**T:**
Look at the definition carefully and try again.

**T:**
Correct—101 and 205 are not consecutive words.

Are all words in a computer the same length? (If in doubt answer maybe.)

**T:**
Actually it depends upon the machine. But...

For the purposes of this discussion we will assume a "fixed word length computer", that is, a computer in which all words are a given fixed length. For our purposes it does not matter what the length is, or even whether the machine is decimal or binary.

**T:**
What shall we talk about?

It is important to distinguish between the name of a field and the contents of a field. The contents of a field in a node may represent numbers, alphabetic characters, lines, or anything else that the programmer may desire. (Node 1.)

Figure 34. (Continued)
DEFINE IDEL
DEFINE NAME OF A FIELD-
IT IS IMPORTANT TO DISTINGUISH BETWEEN THE NAME OF A FIELD
AND THE CONTENTS OF A FIELD. THE CONTENTS OF A FIELD IN A
NODE MAY REPRESENT NUMBERS, ALPHABETIC CHARACTERS, LINKS, OR
ANYTHING ELSE THAT THE PROGRAMMER MAY DESIRE. (NODE )

DEFINE SEQUENTIAL ALLOCATION-
STORAGE ALLOCATION IS THE SELECTION OF MEMORY LOCATIONS IN
WHICH TO STORE DATA. (ALLOC )

ENTERING MODULE ALLOC ********************

STORAGE ALLOCATION IS THE SELECTION OF MEMORY LOCATIONS IN
WHICH TO STORE DATA.
MEMORY LOCATION *DATA

IN ALLOCATING STORAGE FOR A LINEAR LIST, WE WILL ASSUME THAT
EACH NODE IN THE LINEAR LIST REQUIRE C CONSECUTIVE WORDS
OF MEMORY, WHERE C IS A CONSTANT FOR THE LINEAR LIST-
LINEAR LIST *NODE *CONSECUTIVE WORDS OF MEMORY

TRUE OR FALSE: ALL NODES IN A LINEAR LIST CONTAIN THE SAME
NUMBER OF WORDS.

TRUE
RIGHT

ASSUME THAT YOU ARE ALLOCATING STORAGE FOR A LINEAR LIST IN
WHICH EACH NODE CONSISTS OF 3 COMPUTER WORDS. WHAT IS THE
VALUE OF C FOR THAT LINEAR LIST?

RIGHT

Figure 34. (Continued)
IF C = 2, HOW MANY WORDS OF COMPUTER MEMORY ARE REQUIRED TO CONTAIN A LINEAR LIST WITH AT MOST 25 NODES?

T+50-
CORRECT

THE TWO BASIC TYPES OF STORAGE ALLOCATION ARE SEQUENTIAL ALLOCATION AND LINKED ALLOCATION.

T+*

SEQUENTIAL ALLOCATION FOR LINEAR LISTS IS A TYPE OF STORAGE ALLOCATION IN WHICH THE NODES OF THE LINEAR LIST ARE PLACED IN CONSECUTIVE MEMORY LOCATIONS.

T+RETURN-
LEAVING MODULE ALLOC

DEFINE CONTENTS OF A NODE-
DELETING A NODE FROM THE END OF A LINEAR LIST WITH SEQUENTIAL ALLOCATION ONLY INVOLVES CHANGING THE COUNT OF NODES WITHIN THE LIST OR A POINTER TO THE END OF THE LIST. HOWEVER, DELETING A NODE WITHIN THE LIST INVOLVES MOVING THE CONTENTS OF ALL SUBSEQUENT NODES UP TO DIFFERENT LOCATIONS TO ELIMINATE THE "HOLE" LEFT BY THE DELETED NODE. (DELETE)

DEFINE CONTENTS OF A NODE-
DELETING A NODE FROM THE END OF A LINEAR LIST WITH SEQUENTIAL ALLOCATION ONLY INVOLVES CHANGING THE COUNT OF NODES WITHIN THE LIST OR A POINTER TO THE END OF THE LIST. HOWEVER, DELETING A NODE WITHIN THE LIST INVOLVES MOVING THE CONTENTS OF ALL SUBSEQUENT NODES UP TO DIFFERENT LOCATIONS TO ELIMINATE THE "HOLE" LEFT BY THE DELETED NODE. (DELETE)

ENTERING MODULE DELETE

DELETING A NODE IS AN OPERATION IN WHICH A NODE IS REMOVED FROM A DATA STRUCTURE SUCH AS A LINEAR LIST.

T+*
Although deletions frequently occur at the ends of a linear list, it is sometimes necessary to delete a node within the list. The way in which the deletion is accomplished depends upon the storage allocation used for the list.

ENDS OF A LINEAR LIST *STORAGE ALLOCATION

To delete a node from the end of a linear list with sequential allocation only involves changing the count of nodes within the list or a pointer to the end of the list. However, deleting a node within the list involves moving the contents of all subsequent nodes up to different locations to eliminate the "hole" left by the deleted node.

*SEQUENTIAL ALLOCATION

To delete node X(k) from a linked list, it is necessary to access node X(k-1) and change the pointer field that node X(k-1) now points to the node which previously followed node X(k).

*PREVENT FIELD *LINKED LIST

It is easy to delete an item from within a linked list, but with sequential allocation such a deletion frequently involves moving a large part of the list up to different locations.

LEAVING MODULE DELETE ********************

D*DEFINE INSERT-
INSERTING IS AN OPERATION IN WHICH A NODE IS ENTERED INTO A DATA STRUCTURE SUCH AS A LINEAR LIST. [INSERT ]

D*DEFINE QUEUE-
A QUEUE IS A LINEAR LIST FOR WHICH ALL INSERTIONS ARE MADE AT ONE END OF THE LIST AND ALL DELETIONS (AND USUALLY ALL ACCESSES) ARE MADE AT THE OTHER. [QUEUE ]

ENTERING MODULE QUEUE ********************

A QUEUE IS A LINEAR LIST FOR WHICH ALL INSERTIONS ARE MADE AT ONE END OF THE LIST AND ALL DELETIONS (AND USUALLY ALL ACCESSES) ARE MADE AT THE OTHER.

*LINEAR LIST *ACCESSES INSERTIONS AND DELETIONS

Figure 34. (Continued)
A queue may be thought of as being like a line of people "queued up" or waiting in line at a cafeteria. People join the line at one end of the queue, called the rear, and leave the queue at the other end, called the front.

To:

The ends of a queue in a computer are also called the front and the rear. The node most recently placed in the queue is at the rear of the queue.

To:

At which end of a queue do insertions occur?

To:

Read - front or rear?

To:

Rear - right

To:

At which end of a queue do deletions occur?

To:

Front - correct

If the linear list

\[ X_1, X_2, X_3, X_4, X_5 \]

is being processed as a queue and the last node placed in the queue was \( X_5 \), which node is the next node to be deleted from the queue?

\[ X_1 \]

Good

Queues are sometimes called first-in-first-out or FIFO lists because the item removed from a queue is always the "oldest" item.

To:

Return - leaving module queue ---------------

Figure 34. (Continued)
**DiDErINc A LINCAK LIST**

A LINEAR LIST IS A SET OF N GREATER THAN OR EQUAL TO ZERO NODES

\[ x(1), x(2), \ldots, x(n) \]

WHOSE STRUCTURAL PROPERTIES INVOLVE ONLY THE LINEAR ORDER OF THE NODES. [LLIST]

**DiDO WHAT I WANT AND NOT WHAT I SAY**

EITHER SEQUENTIAL ALLOCATION OR LINKED ALLOCATION MAY BE USED FOR A LINEAR LIST. THE OPERATIONS TO BE PERFORMED ON THE LINEAR LIST SHOULD BE CONSIDERED WHEN THE CHOICE OF STORAGE ALLOCATION IS MADE. [ALLOC]

**DiDEFINe EMPTv LIST**

FOR N EQUAL TO ZERO, THE LINEAR LIST IS SAID TO BE EMPTY. [LLIST]

**DiWHAT IS N**

IT IS IMPORTANT TO CONSIDER THE OPERATIONS PERFORMED ON A LINEAR LIST. [LLIST]

**Di*TuTUR LLIST**

ENTERING MODULE LLIST

A LINEAR LIST IS A SET OF N GREATER THAN OR EQUAL TO ZERO NODES

\[ x(1), x(2), \ldots, x(n) \]

WHOSE STRUCTURAL PROPERTIES INVOLVE ONLY THE LINEAR ORDER OF THE NODES.

*NODE *LINEAR ORDER

**Di**

WHICH IS THE MOST PRECISE DEFINITION OF A LINEAR LIST?

A LINEAR LIST IS...

1. A FINITE SET OF NODES.
2. A TOTALLY ORDERED SET OF NODES.
3. A (POSSIBLY EMPTY) FINITE SET OF NODES WHICH IS TOTALLY ORDERED.

*FINITE SET *TOTALLY ORDERED SET *EMPTY SET

**DiRETURN**

LEAVING MODULE LLIST

**DiDEFINe POP**

PEOPLE FREQUENTLY SAY THAT THEY "POP" A STACK MEANING THAT THEY DELETE THE TOP ITEM, OR THAT THEY "PUSH DOWN" A STACK MEANING THAT THEY PLACE AN ITEM ON THE STACK. [STACK]

**DiSTOP**

GOOD BYE

END GITIT 20-7 SEC.

Figure 34. (Continued)
RUN GIITIT -
RUNNING

**P**

WHAT SHALL WE TALK ABOUT?

**P**

DEFINE POP AGAIN.

PEOPLE FREQUENTLY SAY THAT THEY "POP" A STACK MEANING THAT THEY DELETE THE TOP ITEM, OR THAT THEY "PUSH DOWN" A STACK MEANING THAT THEY PLACE AN ITEM ON THE STACK.  [STACK]

**P**

ENTERING MODULE STACK

A STACK IS A LINEAR LIST FOR WHICH ALL INSERTIONS AND DELETIONS (AND USUALLY ALL ACCESSES) ARE MADE AT THE SAME END OF THE LIST.

*LINEAR LIST *ACCESSES INSERTIONS AND DELETIONS

**T**

A STACK MAY BE THOUGHT OF AS BEING LIKE A STACK OF TRAYS IN A CAFETERIA. WHEN WE REMOVE A TRAY FROM THE STACK WE REMOVE THE TOP TRAY. WHEN WE PLACE A TRAY ON THE STACK WE PLACE IT ON THE TOP OF THE STACK.

**T**

WHEN TALKING ABOUT STACKS IN A COMPUTER WE REFER TO THE ENDS OF THE LIST AS THE TOP AND THE BOTTOM OF THE STACK. THE NODE MOST RECENTLY PLACED ON THE STACK IS SAID TO BE ON THE TOP.

**T**

AT WHICH END OF A STACK DO INSERTIONS AND DELETIONS OCCUR?

**T**

TOP

**T**

RIGHT

MOST ACCESSES OCCUR AT THE TOP OF A STACK, BUT IT IS POSSIBLE TO ACCESS A NODE WITHIN A STACK.

**T**

Figure 35. The Complete Protocol for $S_1$ on Run 3
If the last item placed on the stack
X[1], X[2], X[3], X[4], X[5]
was X[5] and we delete a node, which node will be deleted?

\text{TRUE} X[5] = \text{CORRECT}

Because all insertions and deletions occur at the same end of a stack, the item removed from the stack is always the item most recently placed on the stack. For this reason stacks are sometimes called last-in-first-out or LIFO lists.

\text{TRUE}

Stacks have also been called push-down lists, reversion storages, cellars, nesting stores, piles, and even yo-yo lists.

\text{TRUE}

People frequently say that they "pop" a stack meaning that they delete the top item, or that they "push down" a stack meaning that they place an item on the stack.

\text{LEAVING MODULE STACK} ~~~~~~~~~~~~~

\text{TRUE} YOU SAY THAT ALL INSERTIONS AND DELETIONS OCCUR AT THE SAME END OF THE \text{STACK}.

You say that all insertions and deletions occur at the same end of the stack. But "but I don't believe that's true for a FIFO stack." Inserting a node at the end of a sequentially allocated list requires placing the data in the appropriate node and changing a count of the number of nodes in the list or a pointer to the end of the list, but inserting a node within such a list also involves moving all subsequent nodes in the list down to other memory locations to make room for the node being inserted. (Insert)

\text{DEFINE FIFO-}

Queues are sometimes called first-in-first-out or FIFO lists because the item removed from a queue is always the "oldest" item. (Queue)

\text{ENTERING MODULE QUEUE} ~~~~~~~~~~~~~

Figure 35. (Continued)
A queue is a linear list for which all insertions are made at one end of the list and all deletions (and usually all accesses) are made at the other.

A queue may be thought of as being like a line of people "queued up" or waiting in line at a cafeteria. People join the line at one end of the queue, called the rear, and leave the queue at the other end, called the front.

The ends of a queue in a computer are also called the front and the rear. The node most recently placed in the queue is at the rear of the queue.

At which end of a queue do insertions occur?

CORRECT

At which end of a queue do deletions occur?

CORRECT

If the linear list X[1], X[2], X[3], X[4], X[5] is being processed as a queue and the last node placed in the queue was X[5], which node is the next node to be deleted from the queue?

X[5] (X[1])

Queues are sometimes called first-in-first-out or FIFO lists because the item removed from a queue is always the "oldest" item.

Stop.

Good bye

Figure 35. (Continued)
Table 21. Definitions Given by S

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
<td>A type of computer memory that stores information in such a manner that the method of retrieval is predetermined. Examples of stacks are memories that operate on a last-in, first-out basis and memories that operate on a first-in, first-out basis.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>(Change in rating of above definition.)</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>A type of machine memory that records information in a &quot;top-to-bottom&quot; manner. That is, cells of information can be imagined to be stored as a column or row of information divided into different levels. Associated with the stack is a pointer that allows reference to different levels of the stack. Note: Here &quot;machine&quot; means physical or abstract machine. &quot;Pointer&quot; refers to a word or segment of a word of computer memory that holds the address of the beginning of a block of information. All pointers must be stored in memory, while the pointer of the block of information presently being used may also reside in a register.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1/5</td>
<td>A stack is a linear list in which all changes in the list are made at the top or bottom of the list.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1/12</td>
<td>(Change in rating of above definition.)</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>3/1</td>
<td>(Change in rating of above definition.)</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>3/1</td>
<td>A stack is a linear list in which all changes (and usually all accesses) are made at the ends of the list.</td>
<td>A</td>
</tr>
</tbody>
</table>

"bead"

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
<td>I don't know</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1/9</td>
<td>The same as a node.</td>
<td>A</td>
</tr>
</tbody>
</table>
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>None</td>
<td>Reserving portions of primary or secondary computer memory for some particular block of information.</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>Reserving portions of primary or secondary computer memory for some particular block of information. By &quot;secondary memory&quot; I refer to &quot;on-line&quot; secondary memory. Once some portions of memory have been reserved for some reason, it cannot be used to hold other information until released by the computer via the user or operator.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1/14</td>
<td>(Change in rating.)</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>Refers to accessing information from a stack memory. Depending on the type of memory involved (last in-first on, etc.) only a predetermined portion can be accessed at a time. As one &quot;cell&quot; of information is accessed, a pointer advances to the next cell that can now be accessed.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1/7</td>
<td>(Change in rating.)</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1/7</td>
<td>Refers to deleting rather than accessing the top item of a stack.</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>Has to do with groups of computer words.</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1/1</td>
<td>One or more consecutive words of computer memory.</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>&quot;Last-in, first-out.&quot; This is a type of stack memory in which the last cell of information to be placed on the stack is the first cell that can be accessed from the stack. An associated pointer is maintained to locate the various cells in the stack.</td>
<td>B</td>
</tr>
</tbody>
</table>
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>None</td>
<td>(Change in rating.)</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>&quot;Last-in, first-out&quot; This is a type of stack memory in which a pointer records the relative location of each level of information. The most recently stored cell of information is the first cell that can be accessed. If a cell lower than the &quot;top&quot; cell is to be accessed, the pointer must be adjusted to reference that cell. This means that the pointer must sequentially reference each cell above the desired cell.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1/6</td>
<td>(Change in rating.)</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>1/6</td>
<td>A stack in which the last node to be placed on the stack is the first node that can be accessed.</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>(Change in rating.)</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>None</td>
<td>Last-in, first-out. This is another name for a stack. That is, the last node to be placed on the stack is the first node that can be deleted.</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;push down&quot;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>None</td>
<td>Refers to placing information on a stack memory. When a new cell of information is placed on the stack, the pointer is modified to show the new top of the stack. Therefore, call the other cells are &quot;pushed down.&quot;</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>(Change in rating.)</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>Refers to placing a cell of information on a LIFO stack memory. When a new cell of information is placed on the stack, the pointer is modified to show the new top of the stack. Now, each of the other cells in the stack is moved down one notch in the reference priority.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1/8</td>
<td>(Change in rating.)</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>1/8</td>
<td>In the above definition, change &quot;cell&quot; to &quot;node&quot;</td>
<td>A</td>
</tr>
</tbody>
</table>
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>After 3</td>
<td>(Change in rating.)</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>After 3</td>
<td>Delete LIFO</td>
<td>A</td>
</tr>
</tbody>
</table>

"access"

1 None  To retrieve information from primary or secondary memory. That is, to produce a copy of the information that can be manipulated by the computer without altering the representation of the information at its source; i.e. the "place" from which it was accessed. B

2 Before 1 (Change in rating.) C

2 Before 1 Comment: Change the last word of the above definition to referenced. B

2 1/13 (Change in rating.) C

2 1/13 A desired node is located and used for some purpose. The contents of the node can be changed or not changed. If they are altered . . . A

2 1/15 Delete "if they are altered . . ." A

"empty node"

1 None  I don't know. F

2 1/1 A node that contains zeros or garbage of some kind. A Node cannot be blank; i.e., it must contain some character. A

"field"

2 Before 1 Has to do with divisions of a computer word. F

2 1/2 A portion of a node C
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1/2</td>
<td>A named portion of a node. It does not have to contain an integral number of computer words.</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;fifo&quot;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>&quot;First in-first out&quot; A type of stack memory in which the first cell of information that can be accessed is the first cell of information that was stored. Works similar to a paper towel dispenser.</td>
<td>B</td>
</tr>
<tr>
<td>2</td>
<td>1/12</td>
<td>(Change in rating.)</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1/12</td>
<td>A type of stack memory in which the first node that can be deleted is the first node that was placed in the stack.</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>After 3</td>
<td>(Change in rating.)</td>
<td>B-</td>
</tr>
<tr>
<td>4</td>
<td>After 3</td>
<td>A type of linear list in which the first node that can be deleted is the first node that was placed in the list.</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;record&quot;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>To store information in some particular memory. If the information is stored in primary memory, it is usually stored temporarily. It may be stored for longer periods if it is stored in secondary memory.</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1/15</td>
<td>(Change in rating.)</td>
<td>D</td>
</tr>
<tr>
<td>2</td>
<td>1/16</td>
<td>Same as node.</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;linear list&quot;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>I don't know.</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1/3</td>
<td>A finite set of nodes. The set can be empty. The set is also totally ordered.</td>
<td></td>
</tr>
</tbody>
</table>

"fifo"

"record"

"linear list"
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Before 1</td>
<td>The actual computer address of some stored information.</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1/11</td>
<td>Not defined. The address of a node is the address of its first computer word, not its first field.</td>
<td>A</td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>I don't know.</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1/10</td>
<td>A dummy field is a portion of one computer word used to pad a node; i.e., to make a node contain an integral number of computer words.</td>
<td>A</td>
</tr>
</tbody>
</table>

"address of a field"

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Before 1</td>
<td>Whenever a block of information must be stored in more than one physical segment of memory, each segment will contain a portion of information and a pointer to the machine address of the continuation of the information. This pointer is the link that connects these segments.</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1/15</td>
<td>(Change in rating.)</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>Before 3</td>
<td>(Change in rating.)</td>
<td>B</td>
</tr>
<tr>
<td>4</td>
<td>Before 3</td>
<td>A link is a field of a node that is used to point not to the location of the next node of information given whenever that information is stored using linked allocation.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Before 1</td>
<td>I don't know.</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1/5</td>
<td>The beginning or end of a linear list.</td>
<td>A</td>
</tr>
</tbody>
</table>

"link"

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Before 2</td>
<td>To remove one or more nodes from a data structure.</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2/7</td>
<td>(Change in rating.)</td>
<td>A-</td>
</tr>
</tbody>
</table>

"delete"
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2/7</td>
<td>To remove one or more nodes from a data structure. The method used to delete these nodes depends on the type of storage used.</td>
<td>A</td>
</tr>
</tbody>
</table>

"linked list"

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Before 2</td>
<td>A method of storing information that cannot be held in one physical record of computer memory. Part of each record holds some of the information and the rest of the record contains a key that is the address of the first node of the next segment of the data.</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2/2</td>
<td>(Change in rating.)</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>2/2</td>
<td>A linear list that uses linked allocation for storage purposes.</td>
<td>A</td>
</tr>
</tbody>
</table>

"sequential allocation"

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Before 2</td>
<td>A method of storing information such that the information is stored in consecutive physical records.</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2/6</td>
<td>(Change in rating.)</td>
<td>C-</td>
</tr>
<tr>
<td>3</td>
<td>2/6</td>
<td>A method of assigning storage location such that the information is stored in consecutive memory location.</td>
<td>A</td>
</tr>
</tbody>
</table>

"name of a field"

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Before 2</td>
<td>I don't know.</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>2/5</td>
<td>Each field of a node has a name. For example, a field of a node could contain the name of a credit card holder. This is not to be confused with the contents of the field which would actually be that information. If the second field of each node contains a customer's name, the name of that field could be NAME. If someone were named &quot;Smith&quot;, then the contents of the field would be SMITH.</td>
<td>A</td>
</tr>
</tbody>
</table>
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definition</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Before</td>
<td>&quot;empty list&quot;</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>A list in which each node is empty; that is, each node contains garbage.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2/10</td>
<td>(Change in rating.)</td>
<td>D</td>
</tr>
<tr>
<td>3</td>
<td>2/10</td>
<td>A list that contains zero nodes.</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>Before</td>
<td>&quot;link field&quot;</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The field in a node that contains the address of the first node of the next record that follows in a linked list.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2/3</td>
<td>(Change in rating.)</td>
<td>A-</td>
</tr>
<tr>
<td>3</td>
<td>2/3</td>
<td>(Change in rating of above definition made after writing the following definition.)</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2/3</td>
<td>The field in each node of a list using linked allocation that points to the next node.</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>Before</td>
<td>&quot;node&quot;</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>The information stored in the fields of a node.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2/7</td>
<td>(Change in rating.)</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>Before</td>
<td>&quot;insert&quot;</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>To place new node(s) of information into a previously existing list. The node(s) may be placed at the beginning, end, or in the interior of the list.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2/8</td>
<td>(Change in rating.)</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>Before</td>
<td>&quot;linked allocation&quot;</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>A method of assigning computer storage such that each physical record containing nodes of information also holds the address of the record that contains the logically following information.</td>
<td></td>
</tr>
</tbody>
</table>
Table 21. (Continued)

<table>
<thead>
<tr>
<th>Day</th>
<th>Run/Stop</th>
<th>Definitions</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2/1</td>
<td>(Change in rating.)</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2/1</td>
<td>A method of assigning computer storage such that each node contains a pointer to the next node of related information.</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>Before 2</td>
<td>A first in-first out stack.</td>
<td>B</td>
</tr>
<tr>
<td>3</td>
<td>2/9</td>
<td>(Change in rating.)</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>3/2</td>
<td>(Change in rating.)</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>3/2</td>
<td>A linear list in which the first node placed in the list is the first node that can be deleted. It works similar to a paper cup dispenser.</td>
<td>A</td>
</tr>
<tr>
<td>3</td>
<td>Before 2</td>
<td>I don't know.</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>2/7</td>
<td>Nodes do not have names.</td>
<td>A</td>
</tr>
</tbody>
</table>
APPENDIX C

THE DATA STRUCTURE FOR GITIT

The data structure for GITIT consists of a collection of instructional programs, called PI modules, and a script. Both an instructional program and a script can be viewed as programs, and it is the purpose of this appendix to define the languages in which these programs are written.

It should be stressed that the programs involved are two entirely different types of programs. Locus of control is with the machine during the execution of an instructional program, therefore the program must specify the order of presentation of material. Thus the statements in an instructional program are executed in the order in which they occur in the program except when a branch is encountered. However, locus of control is with the student when the script is executed, therefore the order in which script elements are executed is dependent upon the order of the student's statements and questions rather than upon their order within the script.

In order to define the syntax of the data structure as precisely as possible a metalinguistic description of the syntax of an instructional program and a script are given in this appendix. The meta-language used here was developed by John Backus and Peter Naur and is generally called Backus Naur form or BNF (Naur, 1960). BNF has been modified in various ways to account for different features in the
languages described. We will briefly state the conventions used here. The reader is referred to Sterling and Pollack (Sterling, 1970) for a more complete description of BNF.

1. A sequence of characters enclosed in corner braces "< >" represents a metalinguistic variable whose value is given by a metalinguistic formula.

2. The metasymbol "::=" means "is defined as" and is used to separate the metalinguistic variable on the left from its definition on the right.

3. The metasymbol "|" means "or" and separates multiple definitions of a metalinguistic variable.

4. The metasymbol "[X]_i^j" means that X is to be repeated at least i times but not more than j times, where X may be any BNF expression.

5. Braces "[ ]" are used to enclose the English language definition of a metalinguistic variable which cannot be defined by a metalinguistic formula.

6. Any mark or symbol in a metalinguistic formula other than the above metalinguistic symbols represents itself and is an element of the object language being described.

For example, consider the following BNF statement:

<DF command>::=<space>DF<space><text>| <DF command><continuation>

Here the metalinguistic variable <DF command> is recursively defined as either
The Definition of an Instructional Program

Each instructional program or PI module is stored on the disk in 80 character records. The format of each record is as shown in Table 22.

Table 22. The Record Format for PI Modules

<table>
<thead>
<tr>
<th>Position</th>
<th>Description of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>blank</td>
</tr>
<tr>
<td>2-3</td>
<td>operation code or blank</td>
</tr>
<tr>
<td>4</td>
<td>blank</td>
</tr>
<tr>
<td>5-63</td>
<td>text</td>
</tr>
<tr>
<td>65-72</td>
<td>reserved</td>
</tr>
<tr>
<td>73-80</td>
<td>sequence number</td>
</tr>
</tbody>
</table>

The semantics of an instruction program are given in Chapter VI. A formal definition of the syntax of an instructional program is given below.
A Formal Definition of the Instructional Program Syntax

\[
\text{<instructional program> ::= <frame>|<frame><instructional program>}
\]

\[
\text{<frame> ::= <labeled frame>|<unlabeled frame>}
\]

\[
\text{<labeled frame> ::= <label><unlabeled frame>}
\]

\[
\text{<label> ::= <LA command>}
\]

\[
\text{<LA command> ::= <space>LA<space><label word>[<space>]}
\]

\[
\text{<unlabeled frame> ::= <definition>|<statement>|<question>|<submodule call>|<forward pointer>}
\]

\[
\text{<definition> ::= <DF command><backward pointer>}
\]

\[
\text{<DF command> ::= <space>DF<space><text>|<DF command><continuation>}
\]

\[
\text{<statement> ::= <RD command><backward pointer>}
\]

\[
\text{<RD command> ::= <space>RD<space><text>|<RD command><continuation>}
\]

\[
\text{<question> ::= <QU command><backward pointer><answer>}
\]

\[
\text{<QU command> ::= <space>QU<space><text>|<QU command><continuation>}
\]

\[
\text{<answer> ::= <correct answer><wrong answer>}
\]

\[
\text{<correct answer> ::= <correct answer set>|<correct answer set>}
\]

\[
\text{<correct answer> ::= <correct answer>}
\]

\[
\text{<correct answer set> ::= <CA command><alternate correct answer>}
\]
\[ \text{<CA command>::=}<\text{space}>CA<\text{space}><text> \]

\[ \text{<alternate correct answer>::=}<\text{empty}>|<\text{CB command}<\text{alternate correct answer}> \]

\[ \text{<CB command>::=}<\text{space}>CB<\text{space}><text>|<\text{CB command}<\text{continuation}> \]

\[ \text{<wrong answer>::=}<\text{wrong answer set}<\text{unanticipated answer}> \]

\[ \text{<wrong answer set>::=}<\text{empty}>|<\text{WA command}<\text{alternate correct answer}> \]

\[ \text{<WA command>::=}<\text{space}>WA<\text{space}><text> \]

\[ \text{<alternate wrong answer>::=}<\text{empty}>|<\text{WB command}<\text{alternate wrong answer}> \]

\[ \text{<WB command>::=}<\text{space}>WB<\text{space}><text> \]

\[ \text{<unspecified answer>::=}<\text{UN command}<\text{action}>|<\text{UN command}<\text{action}> \]

\[ \text{<unspecified answer}>\]

\[ \text{<UN command>::=}<\text{space}>UN<\text{space}><text> \]

\[ \text{<action>::=}<\text{empty}|<\text{type}|<\text{branch}|<\text{type}<\text{branch}> \]

\[ \text{<type>::=}<\text{TY command}> \]

\[ \text{<TY command>::=}<\text{space}>TY<\text{space}><text>|<\text{TY command}<\text{continuation}> \]

\[ \text{<branch>::=}<\text{BR command}> \]

\[ \text{<BR command>::=}<\text{space}>BR<\text{space}><label word>[<\text{space}>]_{53}^{53} \]

\[ \text{<backward pointer>::=}<\text{BP command}|<\text{BP command}<\text{backward pointer}> \]
<BP command>::=<text>BP<space><text>

<submodule call>::=<CL command>

<CL command>::=<text>CL<space><module name>[<space>]n=53

<module name>::=<letter>[<letter>|<digit>]n=6[<space>]n=59-n

<label word>::=[<character>]7

<continuation>::=[<space>]4<text>

<text>::=[<character>]60

<character>::=<nonblank character>|<space>

<nonblank character>::=<letter>|<digit>|<special character>|<question
mark generator>

<letter>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<digit>::=0|1|2|3|4|5|6|7|8|9

<special character>::=.|[]|(|)|*|+|-|/|\|;|:|,|#|$|%|@

<question mark generator>::=>

<space>::={single unit of horizontal spacing which is blank}

<empty>::={the null string of characters}
The Definition of the Script

The script is a list structure stored on disk in 80 character records, positions 1-72 being used for the script and positions 73-80 for sequence numbers. Lists may be continued from one record to another, as required, in free field format.

The script format presently in use was adapted from the script used for the ELIZA program developed by Weizenbaum (Weizenbaum, 1965) and programmed in GTL by Fricks (Fricks, 1970). The BNF descriptions of the script syntax and the following explanation of the semantics of the script are partially due to Fricks.

A script is composed of one or more script elements. There are three types of script elements: keyword elements, D-list elements and translate elements.

A keyword element associates with each keyword a priority, a keyword equate part (optional), and a composition part. The priority was explained in Chapter VI. The keyword equate part, if used, specifies that every occurrence of the keyword in an input string is to be replaced by the word or words in the keyword equate list. The composition part consists of either a reference element or a list of composition rules.

If the composition part of the keyword element is a reference element, the rules associated with the reference keyword also become associated with the new keyword. In this case, the reference keyword must have appeared earlier in the script as a keyword in a keyword element.
Each composition rule is composed of a decomposition rule and one or more recomposition rules. Four types of decomposition rule elements may be used to determine how the input string is to be decomposed. A decomposition rule word causes the occurrence of that word in the input string to be isolated as a separate segment. A multi-word or D=list part will isolate a word if that word appears in the multi-word list or has been assigned to the D-list designator. A filler part will cause a segment to be formed from any number of words in the input text, even no words at all.

Once the input text has been matched with a decomposition rule, the first element of the associated list of recomposition rules is selected, and the selected recomposition rule is then placed at the end of the list. This causes the machine to cycle through the rules, repeating the first rule after all rules have been used.

If the element selected is a reference element, the decomposition process is restarted, with the rules associated with the reference keyword being used. The reference keyword need not appear earlier in the script when a reference element appears in a recomposition list.

If the element selected is a module reference the action is dependent upon whether or not the mode control switch is on. If the switch is on a module reference causes the machine to shift to tutorial mode to administer the specified module, otherwise it causes the text of the first frame in the module to be printed as a response.

If the element selected is a recomposition rule, a response is generated according to that rule. Most of the recomposition rules in
the present script consist of frame references; however the full capability of the script will be described here.

A frame reference causes the text of the specified frame, followed, in brackets, by the module name, to be placed in the reply string. The appearance of a recomposition rule word causes that word to be printed. A segment designator causes the indicated segment to be printed. A period, comma or question mark designator will cause those symbols to be printed and a special character will itself be printed in the reply.

The second type of script element is the D-list element. The D-list element assigns the D-list word to the list associated with the D-list designator. Any number of words may be assigned to a D-list designator. A keyword may be a D-list word, but its use as such must appear before its use as a keyword in a keyword element. A keyword may not be used as a D-list designator.

The only use of the D-list element in the present system is in connection with the memory mechanism as explained in Chapter VI. D-list elements may also be used whenever it is desirable to assign a D-list designator to any number of words, and then use the D-list designator in a decomposition rule in which case it will match any of the words assigned to it.

The third type of script element is the translate element. This element causes every occurrence of the input word in the input text to be replaced by the word or words appearing in the output part of the translate element.
Every script must contain 'N0NE' as a keyword entry. When the keystack is empty and the memory mechanism is not in the proper state to produce a reply, the composition rules associated with 'N0NE' are applied.

Another special keyword entry is 'MEMRULE,' which is used in conjunction with the D-list designator 'MEM.' Keywords are assigned to the 'MEM' D-list by use of D-list elements in the script.

Each time a keyword is selected from the keystack, a check is made to see if it belongs to the 'MEM' D-list. If it does, the composition rules associated with 'MEMRULE' are used to form a reply which is stored for use later. Once this response has been stored, the keyword and input text are handled in the usual manner, with a reply being generated for immediate use. Sometime later, when the keystack is empty and a counter has reached a certain point, the reply stored in memory is retrieved and used as the reply. If the word 'MEM' does not appear in the script as a D-list designator, the memory function will be inoperative.

A formal definition of the script syntax is given below.

A Formal Definition of the Script Syntax

```
<script>::=(<script element>)$|<script element><script>

<script element>::=<keyword element>|<D-list element>|<translate element>

<keyword element>::=<keyword><priority><keyword equate part><composition part>
```
<priority>::=<positive integer>

<keyword equate part>::=<empty>|=<keyword equate word>|=(<keyword equate list>)

<keyword equate word>::=<word>

<keyword equate list>::=<word>|<keyword equate list><word>

<composition part>::=<composition rule list><reference element>

<composition rule list>::=<composition rule>|<composition rule>

<composition rule>::=((<decomposition rule>)<recomposition rule list>)

<decomposition rule>::=<decomposition rule element>|<decomposition rule>

<decomposition rule element>::=<decomposition rule word>|<filler part>|((<D-list part>))|(<multi-word part>)

<decomposition rule word>::=<word>

<filler part>::=0

<D-list part>::=/<D-list designator>

<D-list designator>::=<word>

<multi-word part>::=*<multi-word list>
<multi-word list>::=<word>|<multi-word list><word>

<recomposition rule list>::=(<recomposition rule>)(<recomposition rule list>|(<module element>)|(<module element>)<recomposition rule list>|(<reference element>)|(<reference element><recomposition rule list>

<recomposition rule>::=<recomposition rule element>|<recomposition rule><recomposition rule element>

<recomposition rule element>::=<recomposition rule word>|<segment designator>|<frame designator>|<period designator>|<comma designator>|<question mark designator>|<special character>

<recomposition rule word>::=<word>

<segment designator>::=<positive integer>

<frame designator>::=(@<module name><record number>)

<period designator>::=%

<comma designator>::=;

<question mark designator>::=Q

<special character>::=[|]*+\-|/=|:|#|$|&|%|@

<module element>::=@<module name>

<module name>::=<letter>[<letter>|<digit>]*
<reference element>::=<reference keyword>

<reference keyword>::=<keyword>

<D-list element>::=<D-list word>/<D-list designator>

<D-list word>::=<word>

<D-list designator>::=<word>

<translate element>::=<input word>=<output part>

<input part>::=<word>

<output part>::=<word>|<output part><word>

<keyword>::=<word>

<word>::=<letter>|[<letter>|<digit>]<space>|<word><space>

<positive integer>::=<digit>|<positive integer><digit>

<letter>::=A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<digit>::=0|1|2|3|4|5|6|7|8|9

<space>::={single unit of horizontal spacing which is blank}

<empty>::={the null string of characters}
This appendix is included to provide documentation for the computer program.

A Brief Description of the Program

The computer program is written in GTL, an experimental programming language developed by the Rich Electronic Computer Center at Georgia Tech (GTL, 1971). GTL is a superset of Burroughs Extended ALGOL and includes extensive string processing capability as well as a non-standard version of LISP 2, a list processing language.

GTL, like ALGOL, is a recursive programming language, thus permitting a procedure to call itself. The GITIT program makes extensive use of recursive programming techniques. There are three major procedures: CONVERSE, PROGINST, and CONTROL. The procedure CONTROL is indirectly a sub-routine of both CONVERSE and PROGINST, and it contains both CONVERSE and PROGINST as sub-routines. Thus CONVERSE and PROGINST can call each other and themselves to arbitrary depths of recursion.

After certain initialization is performed, the procedure CONVERSE is called. Then, whenever the control command *TUTOR is given, or the machine instigates a shift to tutorial mode, or the tutorial mode routine calls a sub-module, the procedure PROGINST is called. Whenever the command *DISCUSS is given, the procedure CONVERSE is called. The
command *RETURN causes an exit from either CONVERSE or PROGINST, and the
completion of a module or sub-module in tutorial mode causes an exit
from PROGINST and a return to the calling routine. The return linkage
is handled automatically by the B-5500 operating system.

The limiting factor for the depth of recursion is not a program
parameter but the B-5500 stack. The actual limit is unknown, but does
not appear to be of practical significance. In one test run, 12 modules
were entered and suspended without encountering a stack overflow on the
machine. However, the student becomes confused after at most four or
five levels of recursion. Therefore, the student may call our the pro-
cedures to an arbitrary depth limited by his own capacity to retain
"return linkage," and modules may call other modules to any required
depth.

The data structure for the discussion mode, the script, is a list
structure. The discussion mode routine, which was adapted from a GTL
program written by Fricks (Fricks, 1970), uses the GTL LISP system.

When the program is initiated, a check is made to see if there
is a file named "RGITIT/usercode" where usercode is the user code of the
student. This file is the "remember" file for the student and will not
be available the first time a student uses the program.

If the remember file is not on disk, the script is read into core
from the file "SCRIPT/usercode." When the program is terminated, the
internal representation of the script is written on the remember file.
When the student uses the program again the script is read from the
remember file and the SCRIPT file is not used.
Each script element is a list such that the first element of the list is an atom, either a keyword, a D-list word or an input word. In the internal representation of the script, the remainder of the list is stored as the property list of that atom. Thus when a word is encountered in the input string, the program can determine whether or not the word is a keyword by examining its property list. GTL provides a hashing routine for storing atoms. It is, therefore, unnecessary to search through tables to recognize keywords or locate the script element for a keyword, D-list word or input word.

The PI modules, which are the data structure for the tutorial mode, are stored on the B-5500 disk as 80 character fixed format records. In the present implementation, each module is a file. When a module is selected the file named

\[
\text{modulename/usercode}
\]

is opened and read sequentially. When a branch is executed a space command is used to reposition the reading mechanism. Whenever a module is suspended to administer another module, the record number is stored and the file closed. When the module is resumed, the file is opened and repositioned to the correct record.

The Control Commands

Whenever a student input begins with an \# , the procedure CONTROL is called. The commands recognized by the CONTROL procedure are given in Table 23. The major commands were discussed in Chapter VI; the additional commands are primarily of interest to the programmer.
Table 23. Control Commands Recognized by the GITIT System

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>*DISCUSS</td>
<td>Enter discussion mode</td>
</tr>
<tr>
<td>*TUTOR</td>
<td>Enter tutorial mode</td>
</tr>
<tr>
<td>*RETURN</td>
<td>Return from discussion or tutorial mode</td>
</tr>
<tr>
<td>*HELP</td>
<td>Explain commands</td>
</tr>
<tr>
<td>*START OVER</td>
<td>Reinitialize the program</td>
</tr>
<tr>
<td>*STOP</td>
<td>Terminate the program</td>
</tr>
<tr>
<td>*CONTROL</td>
<td>Turn on mode control switch</td>
</tr>
<tr>
<td>*CONTROL OFF</td>
<td>Turn off mode control switch</td>
</tr>
<tr>
<td>*DEBUG</td>
<td>Turn on debugging switch</td>
</tr>
<tr>
<td>*DEBUG OFF</td>
<td>Turn off debugging switch</td>
</tr>
<tr>
<td>*WHERE</td>
<td>Specify mode and module</td>
</tr>
<tr>
<td>*MORE</td>
<td>Extend time limit (not used at present)</td>
</tr>
<tr>
<td>*USAGE</td>
<td>Print number of LISP records in use</td>
</tr>
</tbody>
</table>

The commands *DISCUSS and *TUTOR cause CONTROL to call the procedures PROGINST and CONVERSE, respectively. *RETURN causes a return from CONTROL and must also be recognized by the procedures PROGINST and CONVERSE to cause a return to CONTROL and thus to the previous procedure.

The command *HELP causes the machine to administer a special PL module named "XPLAIN." The object of this module is to explain the various control commands.

The command *START OVER causes the machine to reinitialize the program, reading in a new copy of the script, terminating all suspended procedures, and then calling the procedure CONVERSE. The command *STOP
is used to terminate the program.

After processing each of the commands discussed below, control returns to the calling routine which then expects another student input.

The commands *CONTROL and *CONTROL OFF are used to set the Boolean variable CONTROLLING to the values of true and false. The effect of this mode control switch is explained elsewhere. Similarly, *DEBUG and *DEBUG OFF are used to set the variable DEBUGGING to the values true and false. When the debugging switch is on, a message is printed whenever a response is placed on the memory stack in the discussion mode and whenever a submodule is called by a CL command in the tutorial mode.

The *WHERE command causes the machine to state the current mode and the module which is active. The *MORE command is not meaningful in the present version. *USAGE causes the machine to respond with the number of LISP records in use.

The CONTROL procedure thus provides for switching between the discussion and tutorial modes and services various requests.

A Listing of the Computer Program

The following is a listing of the computer program GITIT. The reader is referred to the GTL reference manual (GTL, 1972) for a description of the GTL language.
THE GEORGIA INSTITUTE OF TECHNOLOGY INSTRUCTIONAL TRANSLATOR

GITIT (as a program) is an experimental computer system which can operate in
a machine-controlled mode or in a tutorial
mode. Commands are provided by which the
student can cause the program to shift modes.

GITIT was written in GLT, an experimental programming language
developed at the Georgia Institute of Technology for the Burroughs B-5500.

GITIT was written by Margaret E. Dexter
April 1972

The routines for the discussion mode
were adapted from the ELIZA program
written by Charles L. Franks
June 1970

The remember-recall and timeout routines
were added to ELIZA by John Covert
February 1971

275
8200  \textbf{I=LENGTH(S)*TIME(*)/63)}
8300  \textbf{FOR T IN S DO}
8400  \textbf{IF I=I-1 AND T = 0 THEN RETURN T1}
8500 \textbf{END OF RANDOM}
8600 \textbf{x=x*x}
8700 \textbf{z=x*x}
8800 \textbf{END OF SEARCH}
8900 \textbf{BOOLEAN PROCEDURE SEARCHIT}
9000 \textbf{BEGIN}
9100 \textbf{SEARCH(SC,SEA[0])}
9200 \textbf{IF SFA[0] GTR 0 THEN RETURN TRUE}
9300 \textbf{IF SFA[0] = 0 THEN}
9400 \textbf{PRINT \textquoteleft INVALID USER OF SCRIPT FILE\textquoteright}
9500 \textbf{ELSE}
9600 \textbf{PRINT \textquoteleft SCRIPT FILE NOT ON DISK\textquoteright}
9700 \textbf{EXIT FALSE}
9800 \textbf{END OF SEARCH}
9900 \textbf{x=x*x}
10000 \textbf{END OF SEARCH}
10100 \textbf{BOOLEAN PROCEDURE ASSEMBLY}
10200 \textbf{BEGIN}
10300 \textbf{LABEL L1, L2, L3, L6, P1, ASSEMBLY}
10400 \textbf{STRING MODULENAME(T)}
10500 \textbf{NEXT L1}
10600 \textbf{IF NULL(PUNI=COR(POINT)) THEN RETURN FALSE}
10700 \textbf{STEP 1} \textbf{FOR T=1 STEP 1 UNTIL MAXSEG DO TABL(I)=0}
10800 \textbf{I=0}
10900 \textbf{R=1}
11000 \textbf{L=1}
11100 \textbf{IF CAR(NEXT) = "0" THEN BEGIN}
11200 \textbf{IF NULL(CUR(NEXT)) THEN BEGIN}
11300 \textbf{TABL(I)=I+1=I}
11400 \textbf{GO TO ASSEMBLY}
11500 \textbf{END}
11600 \textbf{P=1}
11700 \textbf{IF ATRM(CAR(NEXT)) THEN BEGIN}
11800 \textbf{IF CAR(NEXT)=CAR(R) THEN BEGIN}
11900 \textbf{NEXT=CAR(NEXT)}
12000 \textbf{RETI TO LIJ}
12100 \textbf{ELSE GO TO L61}
12200 \textbf{DL=IF CAR(NEXT)="0" THEN GO TO ASSEMBLY}
12300 \textbf{ELSE GO TO ASSEMBLY}
12400 \textbf{NEXT=CAR(NEXT)}
12500 \textbf{L61 TABL(I)=NCONC(TABL(Y),CONS(CAR(R)|0))}
12600 \textbf{NEXT=CAR(NEXT)}
12700 \textbf{ELSE GO TO LIJ}
12800 \textbf{END}
12900 \textbf{IF ATRM(CAR(NEXT)) THEN BEGIN}
13000 \textbf{IF CAR(R)=CAR(NEXT) THEN TABL(I)=I+1=CONS(CAR(R)|0)}
13100 \textbf{ELSE GO TO LIJ}
13200 \textbf{ELSE BEGIN}
13300 \textbf{DL=IF CAR(NEXT)="0" THEN GO TO ASSEMBLY}
13400 \textbf{ELSE GO TO LIJ}
13500 \textbf{TABL(I)=I+1=CONS(CAR(R)|0)}
13600 \textbf{ELSE GO TO LIJ}
13700 \textbf{IF NULL(NEXT=CUR(NEXT)) THEN BEGIN}
13800 \textbf{IF NULL(R=CUR(R)) THEN BEGIN}
13900 \textbf{ELSE GO TO LIJ}
14000 \textbf{ELSE GO TO LIJ}
14100 \textbf{IF CAR(NEXT)="0" THEN BEGIN}
14200 \textbf{IF NULL(R=CUR(R)) THEN BEGIN}
14300 \textbf{ELSE GO TO LIJ}
14400 \textbf{ELSE GO TO LIJ}
14500 \textbf{END}
14600 \textbf{IF NULL(R=CUR(R)) THEN GO TO ASSEMBLY}
14700 \textbf{END}
14800 \textbf{END}
14900 \textbf{END}
15000 \textbf{END}
15100 \textbf{REFERENCE}
15200 \textbf{MODULNAME=CAR(POINT)}
15300 \textbf{CAR(POINT)=1x=(MODULNAME)}
15400 \textbf{START=CAR(R)}
15500 \textbf{GO TO ISF}
15600 \textbf{IF CAR(R)="0" THEN BEGIN}
15700 \textbf{MODULNAME=SPACE}
15800 \textbf{MODULNAME=CAR(R)}
15900 \textbf{PRINT \textquoteleft LET ME TUTOR YOU ON THAT\textquoteright}
16000 \textbf{TUTOR(MODULNAME)}
16100 \textbf{REPLU=\textquoteleft WHAT DO YOU WANT TO DISCUSS NEXT Q\textquoteright}
16200 \textbf{RETURN TRUE END}
16300 \textbf{ELSE RETURN FALSE END}
16400 \textbf{ELSE BEGIN REPLU=CONS(R|0)}
16500 \textbf{RETURN TRUE END}
REPLY := 0
FOR T := 1 STEP 1 UNTIL MAXSEG DO
  FOR T ON TARP(1) DO
    IF CAR(T) = "1" THEN CAR(T) := "0"*
    IF NOT NUMBERP(CAR(T)) THEN REPLY := NCONC(REPLY, CONS(CAR(T), 0))
  END OF ASSEMBLE
END OF PROCEDURE FORMJ

BEGIN
  STRING MODULENAME($)
  COUNT := COUNT + 1
  IF NULL(KEYSTACK) THEN
    IF COUNT = 1 AND NOT NULL(HEMLIST) THEN BEGIN
      REPLY := CAR(HEMLIST)
      KEYS := CONS("0", T)
    END
    ELSE BEGIN
      GO TO WRITEIT END
  END
  IF ASSEMBLE THEN BEGIN
    NULLCHECK := NULL(HEMLIST) THEN 0 ELSE COUNT / 2
    PLIST := NCONC(HEMLIST, CONS(REPLY, 0))
  END
  IF DEBUGGING THEN PRINT "REPLACING", REPLY
  END
GO TO L1

L1
POINT := CAR(KEYSTACK)
KEYSTACK := CONS("0", KEYSTACK)
IF T IN MEM GO
IF POINT = T THEN BEGIN
  POINT := CONS("0", KEYSTACK)
  IF ASSEMBLE THEN BEGIN
    NULLCHECK := NULL(HEMLIST) THEN BEGIN
      KEYS := CONS("0", KEYS)
    END
    ELSE BEGIN
      GO TO L2 END
  END
END
WRITEIT
IF QUOTE := FALSE THEN
  WHILE TRUE DO BEGIN
    IF NOT ATOM(CAR(REPLY)) THEN GO TO L8
    PHIN CAR(REPLY) := PHIN & 1
    L8
    IF NULL(REPLY) THEN GO TO L4
    IF ATOM(REPLY) THEN BEGIN
      IF CAR(REPLY) = "0" THEN BEGIN
        IF CAR(REPLY) = "0" THEN BEGIN
          IF CAR(REPLY) = "0" THEN BEGIN
            IF CAR(REPLY) = "0" THEN BEGIN
              IF CAR(REPLY) = "0" THEN BEGIN
                IF CAR(REPLY) = "0" THEN BEGIN
                  IF CAR(REPLY) = "0" THEN BEGIN
                    IF CAR(REPLY) = "0" THEN BEGIN
                      IF CAR(REPLY) = "0" THEN BEGIN
                        IF CAR(REPLY) = "0" THEN BEGIN
                          IF CAR(REPLY) = "0" THEN BEGIN
                            IF CAR(REPLY) = "0" THEN BEGIN
                              IF CAR(REPLY) = "0" THEN BEGIN
                                IF CAR(REPLY) = "0" THEN BEGIN
                                  IF CAR(REPLY) = "0" THEN BEGIN
                                    IF CAR(REPLY) = "0" THEN BEGIN
                                      IF CAR(REPLY) = "0" THEN BEGIN
                                        IF CAR(REPLY) = "0" THEN BEGIN
                                          IF CAR(REPLY) = "0" THEN BEGIN
                                            IF CAR(REPLY) = "0" THEN BEGIN
                                              IF CAR(REPLY) = "0" THEN BEGIN
                                                IF CAR(REPLY) = "0" THEN BEGIN
                                                  IF CAR(REPLY) = "0" THEN BEGIN
                                                    IF CAR(REPLY) = "0" THEN BEGIN
                                                      IF CAR(REPLY) = "0" THEN BEGIN
                                                        IF CAR(REPLY) = "0" THEN BEGIN
                                                          IF CAR(REPLY) = "0" THEN BEGIN
                                                            IF CAR(REPLY) = "0" THEN BEGIN
                                                              IF CAR(REPLY) = "0" THEN BEGIN
                                                                IF CAR(REPLY) = "0" THEN BEGIN
                                                                  IF CAR(REPLY) = "0" THEN BEGIN
                                                                    IF CAR(REPLY) = "0" THEN BEGIN
                                                                      IF CAR(REPLY) = "0" THEN BEGIN
                                                                        IF CAR(REPLY) = "0" THEN BEGIN
                                                                          IF CAR(REPLY) = "0" THEN BEGIN
                                                                            IF CAR(REPLY) = "0" THEN BEGIN
                                                                              IF CAR(REPLY) = "0" THEN BEGIN
                                                                                IF CAR(REPLY) = "0" THEN BEGIN
                                                                                  IF CAR(REPLY) = "0" THEN BEGIN
                                                                                    IF CAR(REPLY) = "0" THEN BEGIN
                                                                                      IF CAR(REPLY) = "1" THEN BEGIN
                                                                                        IF CAR(REPLY) = "1" THEN BEGIN
                                                                                          IF CAR(REPLY) = "1" THEN BEGIN
                                                                                           IF CAR(REPLY) = "1" THEN BEGIN
                                                                                             IF CAR(REPLY) = "1" THEN BEGIN
                                                                                               IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                 IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                   IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                     IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                       IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                         IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                          IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                            IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                              IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                  IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                    IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                      IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                        IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                          IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                             IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                   IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                 IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                       IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                          IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                              IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                 IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                   IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                      IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                         IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                            IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                              IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                 IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                   IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                     IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                        IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                           IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                              IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                 IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                            IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                               IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                  IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                      IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                         IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                           IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                 IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                   IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                       IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                             IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                               IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                  IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                      IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                          IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                              IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                                  IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                                             IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                                                 IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                                                IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                                                                                                  IF CAR(REPLY) = "1" THEN BEGIN
                                                                                                                                                                                                いろいろにプログラムが実行されるが、テーマが変更されると、利用者が説明するようになる。
PROCEDURE READSTUDENT

READ (AAISTOP], STR)

IF STRT OF AAISTOP) = "T" THEN

BEGIN

MODE := "T"

RETURN

END

RETURN

PART PRINT "WOULD YOU REPEAT THAT PLEASE."

GO TO LI

END OF READSTUDENT

AVAILCOURSE RETURNS TRUE IF COURSE IS ON DISK

PROCEDURE AVAILCOURSE (COURSENAME)

BEGIN

IF COURSEID NOT SPACE THEN BEGIN

CLOSE (COURSEID)

RECNID := *1

END

FILE COURSE WITH REAL (COURSENAME, G) THEN

IF SPACE? LSS 2 THEN RETURN FALSE ELSE RETURN TRUE

END OF AVAILCOURSE
PROCEDURE NEXTLINE;
BEGIN
READ (COURSE, 10, CR, (NO99));
END;

PROCEDURE LENGTH;
BEGIN
INTEGER II,

WHILE (TEXT (II = 1, 1) = SPACE) AND (II GTR 1) DO;
LENGTH = II + 1;
END OF LENGTH;

PROCEDURE MATCH;
BEGIN
MATCH = STR (0, 60) = TEXT;
END;

TYPE;
PROCEDURE TYPE;
BEGIN
PRINT TEXT;
NEXTLINE;
END;

PROCEDURE GETREC positions THE COURSE FILE ON RECORD N
AND RESETS RECORD APPROPRIATELY
PROCEDURE GETREC(N);
BEGIN
INTEGER N;
GETREC (N);
END;

PROCEDURE RESET(COURSENAME, N);
BEGIN
IF AVAILABLE(COURSENAME) THEN
BEGIN
COUSEID = COURSENAME;
GETREC (N); END ELSE;
PRINT "UNABLE TO RESET, COURSENAME, TO REC NO, N;";
END;

PROCEDURE CLOSEFILE;
BEGIN
IF COURSEID REC SPACE THEN
CLOSE (COURSE, RELEASE)); COURSEID = SPACE; RECNO = -1;
END;

TUTOR IS THE DRIVER FOR A PI COURSE IN COURSEWRITER FORMAT
PROCEDURE TUTOR (COURSENAME);
BEGIN
STRING COURSENAME;
END;

BOOLEAN HOLD SUBMODULE;
*BEGIN*

**Example Code:**

```plaintext
BEGIN
  IF OP_CODE = "RD" OR OP_CODE = "DF" THEN GO TO RDJ
  IF OP_CODE = "QU" THEN GO TO QJ
  AN OP_CODE OF "CL" IS A ROUTINE CALL, WHERE THE NAME OF
  THE CALLED ROUTINE IS IN POSITIONS 5-10.

  IF OP_CODE = "CL" THEN
    IF OP_CODE = "FP" AND NOT SUBMODULE THEN
      PRINT OP_CODE; PRINT J; GO TO LIAJ
      END;

    END;

  END;

END.
```

*END*
BEGIN AND OPCODE NEQ "QU" DO
  IF OPCODE = "CA" OR OPCODE = "CB" THEN
    IF MATCH THEN GO TO TESTTYC
  IF OPCODE = "WA" OR OPCODE = "WB" THEN
    IF MATCH THEN GO TO TESTWJ
  IF OPCODE = "UN" THEN GO TO UN;
  NEXTLINE
END;

IF WE GET HERE THERE IS NO UN
  PRINT "TRY AGAIN;"
  GO TO RETRY;
TESTTYC: NEXTLINE; WHILE OPCODE = "CB" DO NEXTLINE;
  IF OPCODE = "TY" THEN TYPE;
  IF OPCODE = "BR" THEN GO TO BR;
  GO TO L1;
TESTWJ: NEXTLINE; WHILE OPCODE = "WB" DO NEXTLINE;
  IF OPCODE = "TY" THEN TYPE;
  IF OPCODE = "BR" THEN GO TO BR;
  GO TO RETRY;
UN1: UN1 = RECNJ;
  IF = COUNTUN THEN
    BEGIN
      COUNTERUN = COUNTUN + 1;
      TYPE;
      IF OPCODE = "BR" THEN GO TO BR;
      GO TO RETRY;
    END;
  NEXTLINE;
  IF OPCODE = "UN" THEN GO TO UN2;
  IF OPCODE = "UN" THEN GO TO UN1;
END;

IF WE GET HERE WE HAVE PASSED THE LAST UN AND WILL REPEAT IT
GETREC(NUN));

THE
RETRY;
GETREC(NCA));
GO TO L2;
FIND THE PROPER BRANCH
LABEL = TEXT(0,6); GETREC(O));
WHILE OPCODE NEQ "99" DO
  BEGIN
    IF OPCODE = "LA" THEN
      IF TEXT(0,6) = LABWORD THEN GO TO L1;
      NEXTLINE;
    END;
  PRINT "UNABLE TO LOCATE LABEL, LABWORD, IN MODULE, COURSENAME;
IN MODULE, COURSENAME;
END;

PROCEDURE SELECT COURSE(COURSENAME);
STRING COURSENAME;
BEGIN
  IF RMARGI GTR 7 THEN
    BEGIN
      IF RMARGI GTR 7 THEN COURSENAME = STR(7,7)
      ELSE COURSENAME = STR(7,7)
    END;
  IF AVALCOURSE(COURSENAME) THEN RETURN;
  L11 COURSENAME := SPACE;
  PRINT "DO YOU KNOW THE MODULE NAME?";
  PRINT "MODULE NAME:
  TYPE;
  READSTU;
  IF STR(0,3) = "NO" THEN
    IF AVALCOURSE(COURSENAME="KOURSE") THEN TUTOR(COURSENAME)
    ELSE PRINT "NEITHER DO I == ASK SOMEONE"
  ELSE IF STR(0,4) NEQ "YES" THEN
    BEGIN
      IF RMARGI GTR 7 THEN COURSENAME = STR(0,RMARGI);
      ELSE COURSENAME = STR(0,7)
      IF AVALCOURSE(COURSENAME) THEN RETURN;
  END;

PROCEDURE PROGINSTR;
    STRING COURSENAME(77);
    SELF(COURSENAME) = TRUE;
    IF COURSENAME = SPACE THEN BEGIN
        PRINT ENTER MODULE NAME,
        PRINT ENTER TUTOR MODULE NAME,
        RETURN;
    END;
    IF MODULE in SPACE THEN BEGIN
        IF RVARGI LSS 7 THEN COURSENAME = STR(0,RVARGI)
        ELSE COURSENAME = STR(7);
    END;
    IF AVAILABLE(COURSENAME) THEN RETURN;
    IF COURSENAME <> NOT AVAILABLE, #1 END;

    BEGIN
        PRINT MODULE NAME,
        PRINT CURROURSENAME,
        PRINT NOT AVAILABLE, 
        RETURN;
    END;

    BEGIN
        PRINT PRGNAME,
        PRINT SELF(COURSENAME) = FALSE;
        PRINT TUTOR(COURSENAME) ELSE,
        PRINT HELP MODE HAS NOT BEEN IMPLEMENTED -- SORRY;
    END;

PROCEDURE EXPLAIN;
    BEGIN
        IF AVAIL(COURSENAME) = "XPLAIN" THEN BEGIN
            PRINT HELP MODE HAS NOT BEEN IMPLEMENTED -- SORRY;
        END;
    END;

PROCEDURE CONTROL;
    BEGIN
        IF STR(1) = "USAGE" THEN BEGIN
            IF STR(1) = "HELP" THEN PRINT "HELP" ON;
            IF STR(1) = "CONTROL" THEN PRINT "CONTROL" ON;
            IF STR(1) = "WHERE" THEN PRINT "WHERE" ON;
            IF STR(1) = "OFF" THEN PRINT "OFF" ON;
            IF STR(1) = "DEBUG" THEN BEGIN DEBUGGING ON;
                IF STR(1) = "CONTROL" THEN BEGIN CONTROLLING ON;
                    IF STR(1) = "WHERE" THEN PRINT "WHERE" ON;
                    IF STR(1) = "OFF" THEN PRINT "OFF" ON;
                    IF STR(1) = "CONTROL" THEN PRINT "CONTROL" ON;
                    IF STR(1) = "WHICH" THEN PRINT "WHICH" ON;
                    IF STR(1) = "EXIT" THEN PRINT "EXIT" ON;
                    IF STR(1) = "START" THEN PRINT "START" ON;
                    IF STR(1) = "END" THEN PRINT "END" ON;
                END;
            END;
        END;
    END;
BEGINNING OF PROGRAM

FILL AA WITH *,*,*,*,*,*,*,*,*; FILL BB WITH *,*,*,*,*,*,*,*,*;

OUTPut(OUT,OUT,20,0,72)

OUTPut(OUT,OUT,20,0,72)

IF #FILE THEN PRINT "FILE NAME: "

OUTPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)

INPUT (OUT, OUT, 20, 0, 72)
APPENDIX E

ALGORITHMS FOR GENERATING THE SCRIPT

As explained in Chapter VII, algorithms have been developed for the automatic generation of the script from the PI modules and the technical vocabulary. The algorithms for generating the script are presented in this Appendix. The reader is referred to Chapter VII for further discussion of the procedure for generating the script.

Script generation is a two-step process. Step 1 consists of the generation of decomposition rules to recognize the terms in the specified technical vocabulary. This is accomplished by Algorithm A. Minor terms must be processed first, then major terms and finally primitive terms.

After the decomposition rules have been generated, the next step involves an automatic indexing procedure which is used to extract responses for the various terms from the PI modules. This is accomplished by applying the decomposition rules to the PI modules as specified in Algorithm B. Algorithm B is applied to all PI modules. If there is a known precedence relation among the modules, the modules should be processed in a recommended order according to the precedence graph. (This is advisable because multiple responses for a given minor term will be placed in the script precisely in the order in which they extracted from the modules, and thus they will be presented to the student in that order.) The special PI modules for primitive terms must
be processed after all other modules.

In the specification of the script elements in the following algorithms, words in upper case represent themselves and words in lower case represent names of items in the script. (See Appendix C for a description of the script syntax.) The steps in Algorithm A are numbered A1, A2, etc. and referred to as 1, 2, etc. within the algorithm, and similarly for Algorithm B. Comments in square brackets explain the function of each step.

Algorithm A
(Generating Decomposition Rules)

This algorithm generates the decomposition rules for the script from the specified technical vocabulary.

A1 [Establish keyword elements for pseudo-keywords "NONE" and "MEMRULE." ] Place the keyword elements "(NONE 1((0)))" and "(MEMRULE 1)" in the script.

A2 [Generate translate elements.] Insert a translate Element "(worda=wordb)" or "(word=(phrase))" in the script for each transformation which is to be performed prior to the decomposition process.

A3 [Perform transformations on term.] Read the next term in the vocabulary. If there are translate elements "(word=phrase)" for any words in the term, perform the indicated transformations.

A4 [Establish the priority of the term.]
Set $P = \begin{cases} 
30 & \text{if a minor term} \\
20 & \text{if a major term} \\
10 & \text{if a primitive term} 
\end{cases}$

A5 [Generate the decomposition rule.] Delete all conjunctions, prepositions and articles from the term yielding the reduced term
\[ t = W_1 W_2 \ldots W_n \]
Then generate the decomposition rule
\[ r = (O W_1 O W_2 O \ldots O W_n) \]

A6 [Determine the keyword.] If any word $W_i$ within the reduced term $t$ is a keyword with priority higher than $P$ then $K = W_i$; otherwise $K = W_i$. Delete "$W_i O$" from $r$.

A7 [Establish new keyword element.] If $K = W_i$ and $W_i$ is not already a keyword then insert the keyword element "(K P)" in the script.

A8 [Insert the decomposition rule.] Insert the composition rule "(r)" under the keyword $K$ preserving the partial order among decomposition rules.

A9 [Establish memory mechanism for major terms.] If the term is a major term then append the reference element "(K/MEM)" to the script and insert the composition rule "(r)" under the pseudo-keyword MEMRULE preserving the partial order among decomposition rules. If all terms have been processed go to step 10. Otherwise go to step 3.
Algorithm B
(Generating Recomposition Rules)

This algorithm generates recomposition rules for the script from the PI modules.

B1  [Insert module reference for major term.] Take the module modulename with the major term majorterm. Insert the module reference "(@ modulename)" in the script as the recomposition rule for majorterm.

B2  [Establish prompt.] Place the recomposition rule

(LETS DISCUSS majorterm %)

with the decomposition rule for the term majorterm under the keyword NONE.

B3  [Process alternate major terms.] Repeat steps 1-2 for all alternate forms of the major term.

B4  [Locate next DF command.] If all DF commands in this module have been processed go to step 6; otherwise take the next DF command and go to step 5.

B5  [Locate minor terms in text.] Apply the translate elements and
the decomposition rules to the text. For all minor terms recognized within the text, insert the frame designator "((@ modulename n))" in the corresponding list of recomposition rules where "n" is the record number of the DF command. Go to step 4.

B6 [Insert Forward Pointer.] If there is an FP command in this module, insert the recomposition rule "((@ modulename n))" with the decomposition rule for the major term under the keyword MEMRULE, where "n" is the record number of the FP command. If all modules have been processed go to step 7; otherwise go to step 1.

B7 [Insert module reference for prerequisite area.] Take the module modulename for the prerequisite area areaname. Insert the module reference "((@ modulename)" in the script as the recomposition rule for areaname.

B8 [Locate next DF command.] If all DF commands in this module have been processed go to step 10; otherwise take the next DF command and go to step 9.

B9 [Locate primitive terms in text.] Apply the translate elements and then the decomposition rules to the text. For all primitive terms recognized within the text, insert the recomposition rule "((@ modulename n))" in the corresponding list of recomposition rules, where "n" is the record number of the DF command. Go to step 8.
B10  [Locate next module.] If all modules for prerequisite areas have been processed go to step 11; otherwise go to step 7.

B11  [Establish NOHIT responses.] Insert the recomposition rules

(I DO NOT QUITE UNDERSTAND %)

(PLEASE REPHRASE THAT %)

(WHAT Q)

(I DIDNT CATCH THAT - PLEASE TRY AGAIN %)

(RUN THAT BY AGAIN %)

under the special keyword NOHIT.

B12  [Eliminate decomposition rules for terms which have no response.] Eliminate all composition rules which have no recomposition rules. Then eliminate all keywords which have no composition rules.
BIBLIOGRAPHY


VITA

Margaret E. Dexter was born in Atlanta, Georgia, on January 28, 1937. She was graduated from North Fulton High School in Atlanta in 1955. She received the degree of B.A. (Mathematics) from Agnes Scott College in 1959 and the degree of M.S. (Information Science) from the Georgia Institute of Technology in 1968.

Miss Dexter was employed by International Business Machine Corporation in Atlanta, Georgia, and Charlotte, North Carolina, from 1959 to 1962, and by J. Ray McDermott, Inc. in New Orleans, Louisiana, from 1963 to 1967. She was employed by Georgia Tech as a Research Associate from 1968 to 1971. She will join the faculty of Augusta College, Augusta, Georgia, as Assistant Professor of Mathematics.