/TUTOR : A Computer-Aided Tutorial in PROLOG/

BY

LISA MARIE WYLIE

B. S., Worcester Polytechnic Institute, 1980

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:

[illegible]
Major Professor

CONTENTS

1. Introduction

TUTOR evolved as an exercise in PROLOG; to study its characteristics, learn its structure and then apply it in the development of an interactive software tool. It was developed to provide an independent, self-paced means to learn the C-PROLOG version of PROLOG. It is a tutorial designed for both new and experienced users and contains a comprehensive set of lessons, summaries and exercises to explain each basic area of the language.

TUTOR is written in C-PROLOG. It presents C-PROLOG in the same structural manner used by Clocksin and Mellish [5] in their book which describes standard PROLOG. The tool must be invoked from within the C-PROLOG interpreter (version 1.4) which is available on the VAX 11/780, Perkin Elmer 8/32 and 3220 or the PLEXUS systems at Kansas State University.

Each requirement for this tool; its intent, why it was developed, who is expected to use it and which topics are/are not covered is described in Chapter 2.

Chapter 3 defines the teaching approach used in the system, the general system features and the particulars of the screen design.

The actual implementation of the system is explained in Chapter 4. Machine portability, file structure, memory restrictions and error handling capabilities are included here along with some of the problems encountered during this stage of the project.

Chapter 5 discusses the testing and evaluation involved in proving the reliability and robustness of the system. How the tool reacts to unexpected input and protects the user from internal difficulties is crucial. User feedback about the features and bugs in the system have helped to improve it and make TUTOR a more effective tool.

Many computer-aided instruction (CAI) tutorials are available today. Chapter 6 examines two of these and compares them to TUTOR. Major similarities and differences found between each tool are described here.

An evaluation of Prolog for this type of application and conclusions drawn from the work done to create a tutorial like TUTOR are discussed in Chapter 7. Extensions to the project are proposed in Chapter 8.

Appendix 1 contains the manual page which is available on each system to briefly describe TUTOR. The "TUTOR:Users Manual" can be found in Appendix 2. The source code that makes up this tool is in Appendix 3.

2. Requirements

Each of the requirements in this chapter was identified and agreed upon during the proposal stage of the project. They were determined by examining the need at Kansas State for a tool of this type and served as the basis for the design.

2.1 Purpose

Presently, the PROLOG language is taught, in conjunction with other programming languages, through formal coursework. A student may also choose to learn PROLOG independently using only a textbook or users guide. The idea of a PROLOG tutorial was conceived to bridge these two modes of learning.

TUTOR is a software tool that provides its users with the necessary guidance and feedback found in a classroom environment. But, unlike in a classroom, the student can control the time and frequency of its use, the duration of each session and the pace at which the material is presented.

2.2 Intended Users

This tool is designed for three types of PROLOG users: the beginner, the intermediate and the experienced. Upon entry into the tool, the user is asked for a skill level. This determines which capabilities the user gets and also the sequence in which lessons are presented. Figure 1 - Basic System Flow, shows how TUTOR handles each of the skill levels described below.

A beginner is assumed to have no previous knowledge of PROLOG. The first lesson gives the user some background about the language. Its origin, history, purpose and various applications are described. The availablity and usage of C-PROLOG in the Kansas State environment are then explained. Once these areas are covered, the user begins learning the language itself. TUTOR automatically guides a beginner through a logical sequence of lessons, summaries and exercises for each major area or topic of the language.

An intermediate user is assumed to have some experience in using PROLOG. By choosing this level, the user is given the choice to continue from any point in a previous session using the tool, or to select a particular topic for review. Essentially, the intermediate level gives the user the ability to start anywhere in the automatic sequence of lessons, setup for the beginner (referred to as "beginner/intermediate"), or to use the tool at the experienced level (referred to as "intermediate/experienced") and choose only desired topics.

An experienced user obviously knows most or all of the language but can use the tool to review any particular

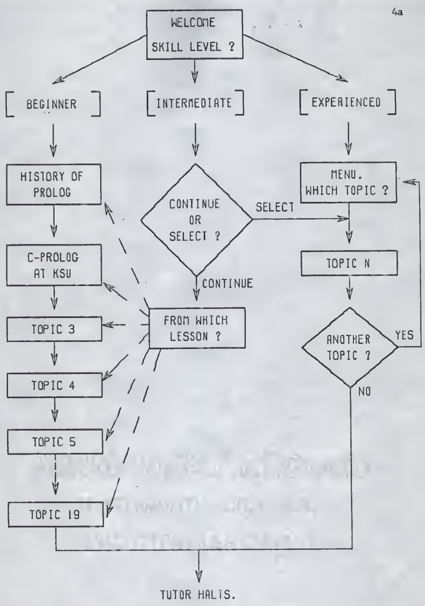

FIGURE 1. BASIC SYSTEM FLOW

topic. A menu is given to the experienced user or the intermediate user who chooses the experienced route, so a topic can be easily chosen. The experienced user is allowed to skip the summary and/or exercise portion of any topic presented.

At the end of every lesson (except the first one), summary and exercise, a beginner or beginner/intermediate user has the option to repeat an earlier topic. When that has been completed, the tutorial will automatically bring the user back to where the repeat was requested. All users are able to leave the tool easily at any time. Before the tutorial and the interpreter are exited, the user is given the chance to change skill levels and continue to use the tool.

2.3 Content

When deciding which topics would make TUTOR a comprehensive tool that would give its user a basic understanding of the C-Prolog language, the table of contents in the Clocksin and Mellish textbook [5] seemed like an appropriate place to start. The first two lessons of this tutorial describe the history of the PROLOG language and the use of C-PROLOG at Kansas State.

Topics 3 through 20 came directly from the table of contents without further investigation at this time.

1) history of PROLOG	7) rules	13) structures
2) C-PROLOG at KSU	8) syntax	14) lists
3) facts	9) characters	15) backtracking
4) questions	10) operators	16) cut
5) variables	11) equality	
6) conjunctions	12) arithmetic	

17) reading/writing characters	19) accessing files
18) reading/writing terms	20) declaring operators

To gain a complete understanding of basic C-PROLOG, it is also necessary to know each of the "core" built-in predicates listed below.

atom()	integer()	reconsult()
atomic()	member()	tab()
consult()	nl	true
fail	nonvar()	var()

A basic explanation of what each predicate does is given even though some of them are also mentioned when describing the basic features of the language.

It is not the intention for TUTOR to cover every area of PROLOG. The following topics are excluded because they are considered to be too advanced for the purposes of this tutorial.

- debugging facilities
- altering stack/memory sizes
- other builtin predicates
- pre-processing
- sets
- nested executions
- internal database
- state file

2.4 Implementation Constraints

TUTOR is written in C-PROLOG and can be used on each machine (VAX 11/780, Perkin Elmer 8/32, 3220 and PLEXUS 1 and 2) that supports Version 1.4 of the language. Its size is purposely small so the tool will run within the interpreter.

The software for this tool is composed of many small files. Each lesson, summary, and exercise is contained in a separate file. The general purpose library, a main routine for each skill level, a routine to ask questions and various other control functions are each contained in a separate file.

Documentation for TUTOR is available in two forms. For each system, there is an on-line manual page which contains most of the "TUTOR:User's Guide". The "TUTOR:Users Guide" is also available in a booklet. It explains how to run the system, what to do when having problems, and supports an extensive glossary of all the PROLOG terminology used in the tutorial.

3. Design

Once the proposal and requirements were complete, the design of TUTOR began. Each requirement was researched and expanded and some were even changed. This chapter describes, in much more detail, what further decisions were made, how the system would work and how C-PROLOG would be taught.

3.1 Teaching Approach

When the design of TUTOR began, it became necessary to define a teaching method or approach to be used throughout the development of the system. Many human factors came into play when deciding how each concept of the language should be conveyed to the user. The system had to be easy to use, robust, and consistent in the way it behaved.

One of the most important qualities of this tutorial is that the pace at which it runs is completely user-controlled. There are no imposed time constraints to read a screen of information or to answer a question. To make it easier to teach PROLOG, the language is broken down into many smaller topics, so each concept is easier to grasp and takes little time to cover.

Each session with the tool can be as short or long as the user chooses. TUTOR can always be invoked at a later date for continuation or review.

Whenever possible, concepts are taught by example - not formalism. It is much easier to understand something if it can be seen. Many examples are displayed throughout the lessons to show the user exactly how the PROLOG interpreter converses with a user.

Almost every PROLOG topic is presented in a lesson/summary/exercise format. The lesson describes the topic, explains its value and gives examples of how it is used. Each of the highlights in the lesson is reviewed in the summary. The exercise has the user repeat what was learned by asking various questions.

During an exercise, the user is asked one-line or multiple-choice questions. This provides a little more variety, but the underlying reasons for using multiple choice questions are twofold. First, questions whose answers are more difficult for the tutorial to interpret, are better handled by a letter choice. There is also less chance for human error when the answer is only a single letter.

For simplicity, the list of choices on a multiple choice question is limited to three. When the user responds with the wrong answer, an explanation is given as to why the answer is wrong and the question with its choices are repeated. One retry is always given if any question is answered wrong the first time. A single-line question is reworded the second time around in case it was misinterpretted originally. Whatever the outcome, the reason for the answer is always supplied.

3.2 General System Features

TUTOR is an interactive system in that it creates a dialog between itself and the user. At the end of every screen and after each question, it expects input. When it receives a complete answer, one ending with a period, the system will interpret the answer and then react. Otherwise, it waits patiently until an answer is entered.

Upon entry into the tool, the initial screen generated is the welcome screen. A brief description of the tool and each of the available skill levels is given. To choose a level, the user must enter 'b.' for beginner, 'i.' for intermediate or 'e.' for experienced. With any other answer, except 'q.' for quit, the system assumes the beginner level and the next screen is printed.

Whenever the next 'n.' option is included in the commands at the bottom of a screen, and the user responds with a command or input not in that set, TUTOR continues to the next screen.

The intermediate level starts off with a menu screen of all the PROLOG topics and a question to find out the desired mode of operation. The beginner/intermediate mode is initiated by asking the tutorial to continue 'c.' from a certain lesson. The intermediate/experienced mode will start when the user types 's.' to review a selected topic. An answer other than 'c.', 's.' or 'q.' , again assumes a continue. The user is then asked to choose the lesson number to review or continue from.

Each time a menu appears on the screen and asks the user to choose a topic, the system expects to read an integer value. Anything except an integer or a 'q.', will repeat the menu.

Another feature for beginner/intermediate users is the repeat 'r.' command. It allows any topic, previous to the current topic being worked on, to be repeated. Repeat is an available command, beginning with the second lesson in the tutorial, at the end of every lesson, summary and exercise.

Anytime the tutorial pauses for input, the commands 'q.', 'quit.', 'exit.' and 'halt.' initiate a departure from the system. Before the session stops, the user has the chance to change skill levels and resume. If a new level is not desired, an 'n.', meaning "no" will stop TUTOR and exit PROLOG completely.

3.3 Screen Design

The medium utilized in teaching PROLOG is the CRT or terminal screen. Every input and output of TUTOR is screen-oriented. To ensure ease in learning, each screen is designed to be simple. The entire width is used for efficiency, but each line of text is double spaced for readibility.

The basis of every screen is from a common outline. A title is always printed in the upper left-hand corner so the user is always aware of which topic is being presented. The quit command appears in the lower right-hand corner of every screen to allow an exit at any time. Continue and repeat commands occupy the bottom line of the screen when they are used.

A common design such as this avoids the problem of having to figure out a new screen each time. More time can be spent concentrating on the material being presented. Instructions of what to do next are always visible.

Menus are used when a beginner chooses to repeat a topic and during the intermediate/experienced and the experienced levels to list the available PROLOG topics for review. If any part of a menu advances off the screen before the user can choose a topic, the system will automatically repeat the entire menu.

4. Implementation

Most of the effort in this project was focused on the implementation phase. This chapter describes each area of the code, the decisions and assumptions that were made and the multiple roadblocks that had to be overcome.

4.1 Tool Structure and Files

The structure of TUTOR consists of many small files. Decided early on, this approach made the programming task much simpler and easier to debug because everything was on a much smaller scale. With limited experience in PROLOG programming, it was easier to study, prepare and program each topic on an individual basis.

Two of the original topics defined during the requirements stage have been removed from the tutorial. They were "structures" and "declaring operators". The subject of structures doesn't contain enough material to warrant inclusion and the process to declare an operator is too advanced for the purposes of TUTOR. It was decided at this time to include the builtin predicates in one lesson at the end of the tutorial. But, like the topics "The History of Prolog" and "C-Prolog at Kansas State", they are only included as extra information for the user and thus do not need a summary and exercise. The predicate "member" is not mentioned in this lesson because it is not part of C-Prolog, only standard Prolog.

The following is a list of each lesson file.

accfile	Lesson #18	Accessing Files
arith	Lesson #12	Arithmetic
backtr	Lesson #14	Backtracking
builtin	Lesson #19	Built-in Predicates
chars	Lesson #9	Characters
conj	Lesson #6	Conjunctions
cpro	Lesson #2	C-PROLOG at KSU
cutt	Lesson #15	Cut
equal	Lesson #11	Equality
facts	Lesson #3	Facts
hist	Lesson #1	PROLOG's history
lists	Lesson #13	Lists
oper	Lesson #10	Operators
quest	Lesson #4	Questions
rules	Lesson #7	Rules
rwchar	Lesson #17	Reading/Writing Characters
rwterm	Lesson #16	Reading/Writing Terms
syntax	Lesson #8	Syntax
vars	Lesson #5	Variables

For each lesson except hist, cpro and builtin, there are corresponding summary and exercise files. All summary file names begin with "s_" and all exercise file names begin with "e_". Each lesson, summary and exercise rule has the same name as the file its contained in.

Other files that essentially control the tutorial are listed below.

ask	- procedure to ask one-line questions
exper	- experienced level screen
get_it	- procedure to present a certain topic
interm	- intermediate level screen
intro	- help screen file
lib	- general purpose library
mult	- procedure for muliple choice questions
redo	- procedure to repeat a topic
repeatt	- allows user to repeat a previous topic
start	- welcome screen
tutor	- file that does initial consulting
Tutor	- executable file to print help screen

4.2 Program Layout

To provide clear and easy-to-read source code, certain guidelines for laying out programs, including some of those suggested by Clocksin and Mellish, were followed.

A group of clauses for a given predicate is called a procedure. Within every file, each procedure begins on a new line and is separated from the next procedure or clause by a blank line. When an entire clause cannot fit on one line, the head of the clause and the first goal are put on the first line and each subsequent goal is indented on a separate line after that.

Each file begins with a comment description of its contents. Whenever a goal is executed and its predicate is defined in clauses outside the file, a comment is added to specify which file the clauses are defined in. This scenario happens in only a few cases.

White space is used consistently to make each file look similar and to make the code more readable.

4.3 Memory

In order for this tool to run on the VAX, 8/32, 3220 and PLEXUS machines, memory constraints were always a consideration. C-PROLOG loaded on the VAX is setup with the following memory sizes:

```
        atom space: 128K (17524 bytes used)
        aux. stack: 8K (0 bytes used)
        trail: 64K (48 bytes used)
        heap: 256K (36876 bytes used)
        global stack: 256K (0 bytes used)
        local stack: 128K (300 bytes used)
```

Memory allocated on the Perkin Elmer 8/32 machine when Prolog is loaded is much more limited as shown below.

```
        atom space: 72K (17524 bytes used)
        aux. stack: 8K (0 bytes used)
        trail: 32K (48 bytes used)
        heap: 72K (40056 bytes used)
        global stack: 72K (0 bytes used)
        local stack: 72K (300 bytes used)
```

Allocated memory on the PLEXUS machines is also different.

```
        atom space: 72K (16892 bytes used)
        aux. stack: 8K (0 bytes used)
        trail: 32K (48 bytes used)
        heap: 72K (36876 bytes used)
        global stack: 72K (0 bytes used)
        local stack: 72K (300 bytes used)
```

At the beginning of the implementation, the code size reached 40K bytes after only a few topics and general purpose procedures were developed. Originally, all files were consulted into the database when the tool was invoked.

A new approach was developed to conserve memory. Only those files which contain clauses needed to do the next few operations are consulted. As soon as a procedure completes and is not needed, it is retracted from the database to make room for another. This method eliminates the initial waiting time, after typing '['/usr/prolog/tutor'].', for every file to be read in. Throughout a session, the user will see periodic messages printed each time a new file is consulted, but will not experience a noticeable time delay.

4.4 Library Functions

A set of common clauses and procedures was developed and is located in a file called "lib". These procedures are used for three basic types of operations: checking, control and output. The library file is consulted each time TUTOR is loaded and each of the procedures it contains remain in the PROLOG database for use by other clauses.

The only checking procedure in the library is called "not_var". When not_var(X,Y) is a goal in the body of a clause, its first argument is instantiated to the immediate user response and the second argument is left a variable.

The not_var procedure then checks the first argument. If it is a variable, the second argument is instantiated to "bad_reply" and control returns to the originating clause. When the user responds with a constant, not_var returns the same constant in the second argument.

Of the six control procedures, "topic" and "repeat_topic" are the simplest. The first one, topic, is used to consult the file that contains the PROLOG topic, invoke the clause that presents the topic and then abolish the clause from the database. Its first argument is instantiated to the name of the topic clause and the second, to the arity of the clause.

Repeat_topic is used each time a dialog procedure prints the "r."epeat command at the bottom of a screen. User input is checked by not_var and then by repeat_topic. If "r." is input, repeat_topic invokes the dialog procedure "repeatt".

A procedure called "seq" has two arguments that define the range of topic numbers to be covered by the tutorial. As long as the arguments remain unequal, the first rule will recursively call "get_it", which presents the next topic. If the arguments become equal, but the tutorial is currently running in the experienced mode, nothing happens. A rule like this allows the intermediate/experienced user to continue using the tutorial after one topic is finished. Otherwise, equal lesson numbers cause the third "seq" rule to print a farewell message and leave the tutorial. The last rule is used to end the beginner or intermediate/beginner sequence.

At the end of every screen, the user must enter input in order to proceed. The clauses "mid_input" and "end_input" were written to check this input, but more importantly, to prevent syntax errors from disrupting TUTOR.

Previously, all input was handled by the "read" predicate. When a syntax error occurred, "read" failed and Prolog dropped out of the rule it was executing. Sometimes it started up another rule or caused the tutorial to abort. These clauses contain a second rule that will always pass if the first rule fails from a syntax error.

The last part of the control category consists of two procedures and a clause that, together, handle the exit feature of this tutorial. As mentioned before, each time TUTOR pauses for input, one option is to quit. The "quit" procedure in the library checks its argument for a "halt" , "q" , "quit" or "exit" entered by the user. If true, "quit" executes the clause "level_stop" which writes 'Do you want to try a new skill level?' to the screen, reads in the answer and executes the procedure "new_lev". If the user answered "y.", new_lev writes a choice of skill levels onto the screen, reads the level chosen, retracts the previous lev from the database and restarts the tutorial at the new level. An "n." or any other answer results in a complete exit from the tutorial and PROLOG.

Lastly, there are eight output clauses that also reside in the library. To make TUTOR screen-oriented, the "newpg" clause was written. It contains 22 calls to the built-in function "nl", or newline, which force the cursor to advance to the top of the screen before the next screen of text is output.

The "bad_inp" clause is used during the intermediate mode when TUTOR expects an integer lesson number, but instead receives a constant. This clause writes the message "INVALID INPUT - TRY AGAIN" to the screen.

During the design, it was decided to always provide the user with visible instructions. With this in mind, the other three output clauses, "next1", "next2" and "next3" were implemented. Respectively, these clauses write out a set of one, two or three commands to the bottom of every screen:

```
TO QUIT - type "q."
NEXT SCREEN - type "n."
TO REPEAT A TOPIC - type "r."
```

Many of the questions in the exercises require a database for the user to work with. The clause "new_db" allows a separate database of four facts to be created at any time.

The facts are asserted into the Prolog database as arguments to the predicates dbl, db2, db3 and db4, after the facts from the last pseudo database are abolished. The fact "show_db", with no arguments, is assserted and used as a flag by the other routines that ask questions.

Whenever a question routine finds the fact "show_db" in Prolog's database, the goal "print_db" outputs the four-fact database before the question is asked. The database is repeated with the question if a retry is needed. The "show_db" fact is retracted when the question is finished.

Lastly, the "menu" clause prints out a menu of all C-Prolog topics, by number, onto the screen. It is used during the intermediate and experienced levels to give the user a selection to choose from.

4.5 General Control Functions

There are eight control functions that are used to run this entire tutorial. The first one, "tutor", is the only file that must be consulted to load the initial files of TUTOR. Negative clauses, those with empty heads, are used in this file to tell Prolog to start proving the goals in its body.

Seven files are consulted as a result and the sentences

```
                To begin, type start.  {RETURN}
                               ----------------
                To quit, type halt.  {RETURN}
                              ---------------
```

are printed on the screen.

A "halt. {RETURN}" will immediately exit the interpreter. If the user types "start. {RETURN}", the "start" clause is invoked. A welcome screen is printed, each of the skill levels is explained and the user is asked to choose one. When a skill is chosen, a fact whose predicate is "c_level" and argument is either "b" or "e" is asserted into the database to keep track of the current skill level the tutorial is running. A rule defining the intermediate skill level is not saved because of the short time that elapses until another mode, equivalent to a beginner or experienced user is chosen. Whenever a skill level changes, this fact is retracted and another put in. pending on the respective skill level, TUTOR will begin the lesson sequence via "control", go to the "interm" rule or use the "exper" rule.

The "interm" rule guides the intermediate level. A menu of topics are presented. If the user chooses to continue from a certain point, the tutorial runs in the beginner mode and the lessons resume with the number chosen. Selecting one topic is also an option, after which TUTOR jumps to the experienced mode. One topic or a set of topics are accessed using the "control" rule in the library. When "interm" receives a non-integer lesson number, it outputs an error message using the "bad_inp" library routine and regenerates a new screen, menu and the question again.

The experienced skill level starts off with the "exper" rule. It outputs the same menu as "interm". Being a recursive rule, "exper" allows the user to select topic numbers until the "q." is received. A single lesson is accessed directly through "get_it" because it is easier and quicker than going through "control". An invalid input will repeat the menu immediately.

In a file called "get_it", there are nineteen "get_it" rules and a "get_it" fact. The rules correspond to the topic number seen in the menu. Each is accessed directly or by the library function "control". The purpose of these "get_it" rules is to: assert a fact containing the current topic number; call the "topic" rule with the name of the lesson, summary or exercise and its arity; and then retract the current topic fact, "c_topic", from the database. The last line in this file contains a "get_it" fact with the anonymous variable as its argument. It succeeds for lesson numbers greater than 19.

All of the exercises use the "ask" or "mult" rules for questions. Single line questions are done using "ask". When this rule starts up, it saves the reworded question (Q2), the answer to the question (Ans) and the reason for the answer (Reas) in the database using "asserta". This predicate was chosen over "assert" because it inserts a fact at the beginning of the database making it quicker to access. Q2 is only used if the first answer is wrong. Once the answer and reason are printed, they are removed from the database. Q2 is also retracted.

Multiple choice questions are output by "mult". The question (Q), the three choices (A, B and C), each reason (RA, RB and RC) and the answer (Ans) are asserted into the database. Whenever one of these is needed, a goal with a variable is used. For example, Ans is put into the database with the goal "asserta(ans(Ans))". Before it is output, the goal "ans(R)" must be satisfied and R gets instantiated to whatever the variable "Ans" was when asserted. Then "R" is used with the "write" predicate. If an answer other than "a,b or c" is given, it is counted as a wrong answer.

The "repeatt" rule is mispelled because its correct spelling is a predicate of Prolog. This rule handles requests to repeat a previous topic. A "r."epeat option is only given to beginners. It uses the "c_topic(X)" fact in the database to determine where the user is in the tutorial. All of the topics that precede "X" appear in a menu. After the topic is chosen and repeated by "get_it", the session picks up from where the user left off.

4.6 Portability

TUTOR was developed on a VAX 11/780 running under System 5 of the UNIX* operating system. It was then ported to the VAX 11/780 (ksuvaxl) at Kansas State that runs under 4.2 BSD UNIX. From there, it was copied onto the PLEXUS systems (ksuplxl and ksuplx2) which run under UNIX System 5, the Perkin Elmer 8/32 (ksu832), and the Perkin Elmer 3220 (ksu3220). Each of these systems contains version 1.4 of the C-Prolog interpreter.

4.7 Assumptions

Appendix 2 of this report contains the TUTOR: Users Guide. It is assumed that this document will be made available by the Computer Science Department to any person who wants to use TUTOR. The manual page will also be on-line on every machine that supports TUTOR.

---

* Trademark of Bell Laboratories

4.8 Limitations

One of the capabilities originally intended for the experienced user was to allow the summary or exercise of a topic to be skipped. This feature could not be implemented in the time allowed.

5. Testing

TUTOR has undergone a series of testing phases throughout its development. As each small piece of the tutorial was coded, its syntax and functionality as a single module was proven. Multiple modules were then brought into the Prolog database for a more integrated series of tests. TUTOR was also tested as a complete system, which includes its documentation, by people with various backgrounds.

5.1 Ongoing Unit Tests

Each function of TUTOR and also every lesson, summary and exercise can be thought of as a single unit. During the implementation phase of this tutorial, every unit was tested for syntax and logic errors as soon as it was written. The Prolog interpreter made this task relatively simple.

Correct syntax of a module was proven if the file could successfully be "consult"ed into the Prolog database. If not, the error was pointed to by the interpreter. After it was corrected, another attempt was made to cons t the file. Once in the database, some functions were tested by using them in questions, varying their arguments and watching the results. Those that contained goals in the library were tested after "lib" was consulted. When an error in the logic was not obvious, the "trace" feature of the debugger was used to do an instruction-by-instruction analysis of the function to see exactly where and how it was failing.

Errors found during unit testing could be avoided in writing the next module. Duplication was also eliminated in many areas by creating general library functions that could be accessed by every module.

5.2 Integration Testing

As development progressed, testing was done on a much broader basis. More and more of the system was tested together and its operation was much more visible.

During this stage, major problems were uncovered. The memory problem, described in section 4.3, made it necessary to "consult" only those files immediately needed by the tutorial and "retract" clauses once they were no longer needed.

A problem with the local stack also appeared. During execution of a larger lesson, the stack ran out of memory because of the method used to write text to the screen. Originally, a recursive clause called "out" was developed to handle an output string. It looked like this:

```
out([]):-nl.
out([ H | []]):- put(H),nl.
out([ H | T ]):- put(H),out(T).
```

The problem was, each letter of text required another call to "out". With every call, the return address was pushed onto the stack and the stack was not cleared until the entire lesson finished, thus causing an overflow condition. To overcome this problem, each "out" and its string in double quotes is replaced with the builtin predicate "write" and the same string in single quotes. The local stack is no longer needed to store return addresses because each "write" completes immediately.

The capabilities of each skill level were examined to make sure each type of user could get to the topics necessary. For the beginner level, the predefined sequence of topics from "Lesson 1 - The History of Prolog" to "Lesson 19 - Builtin Predicates" was verified. TUTOR automatically led the beginner through each of the lessons, summaries and exercises necessary to learn basic Prolog. After the first lesson, the user was always given the option to repeat a previous topic before proceeding to the next section.

A menu is provided to the intermediate and experienced level users. The former can choose to continue from a certain point in the beginner sequence or to select a single topic at a time. Each single topic was accessible from the menu.

5.3 Overall System Testing

When TUTOR was completed, the last phase of testing began. Every section of the tutorial was executed and checked for correctness and completeness. Examples contained in the lessons were tested independently with Prolog.

Expected as well as unexpected responses were tried whenever the system was waiting for input. Some of the unexpected are handled by the code. For instance, whenever a variable is input, it is changed to the constant "bad_reply" and then tested as if it were the answer originally typed in. Some input causes the system to jump into the debugger or output an unfamiliar prompt. When symptoms like these were found, a procedure of what to do was documented in the User's Guide.

The quit "q.", continue "c." and repeat "r." commands were invoked at each place they appear. Changing to a new skill level is allowed at any time by any type of user.

5.4 Evaluation

During all three testing phases, TUTOR was being evaluated. Many types of people saw the system, used it, tested it and shared their opinions. Each requirement was also appraised by the developer.

Overall, the system has consistent external behavior. It is easy to use and always provides the user with prompt and informative responses. The lessons are clear and concise and the exercises test the important points of the topics presented. Explanations are always given for right and wrong answers.

6. Other Tutorials

This chapter describes two other computer-assisted tutorials, "The LISP TUTOR" from Carnegie-Mellon University and "CLYDE: A UNIX TUTOR" developed at Brown University. Comparisons between each of these tools and this PROLOG TUTOR are discussed.

6.1 The LISP TUTOR

Still under development, the LISP TUTOR [1] was written to answer the high demand for courses in LISP, but more importantly, to create a tool that would be as effective as a private tutor. As an alternative or supplement to classroom lectures, it provides intelligent guidance to the student who spends most of the time designing and writing LISP programs.

The LISP TUTOR is designed to run in a transparent mode until the user requests guidance or until it sees a mistake has been made. When difficulties arise during a coding session, it automatically moves the student into a design or planning session and walks them through the algorithm, step-by-step, using an example. The student can then return to writing code with the correct approach still fresh in their mind.

In some cases, the tutor provides hints and reminders or asks questions to get the user back on the right track. It can even go as far as writing a piece of code so the student can proceed. This is only done on request or if the tool decides too many mistakes have been made. This feature prevents wasted time, avoids frustration and leaves little reason for the student to give up, as might have happened if working on the problem alone.

A structured editor provides the tutoring interface for entering code. It automatically balances parentheses and sets up templates for each function. If the student types "(defun" the tutorial will automatically generate

```
(defun <NAME> <PARAMETERS>
       <PROCESS>
)
```

onto the screen. Each area in brackets must be coded. The tutor moves the cursor to the next field that the user is expected to fill in.

Each lesson in the curriculum takes from 1-4 hours to complete. The student uses an instruction booklet with each topic and goes through each of the problems. During a session with the tutor, the display screen is broken up into 3 horizontal windows: a "code window" where the actual code is typed, a "tutoring window" where the tutorial writes its feedback and a "goals window" where the problem definition, reminders or examples are displayed.

The LISP TUTOR is generally accepted by students, but they do complain about the excessive number of menus and its slow response time (actual numbers were not given). It runs under FRANZ LISP, currently on a VAX 725 and requires 3 megabytes of memory for a single user.

6.2 Clyde : A UNIX TUTOR

Clyde [10] is a knowledge-based system that teaches the UNIX operating system to new users. "It is a program which simulates the command level of the operating system, monitoring the session and interrupting the user to offer advice only when warranted." It does its teaching by example.

Like the LISP TUTOR, Clyde remains transparent until the user needs help. It keeps a permanent profile of each user's progress to avoid repetition in a single session and in multiple sessions, just as a human tutor would do.

The tutor covers a variety of topics such as the "history" facility in Berkeley UNIX CShell, the directory heirarchy in UNIX, wildcards and some of the commands in UNIX that can be dangerous to use. It watches each command the user enters and, if possible, suggests a more efficient way to accomplish the same multiple command sequence.

Clyde maintains a wide vocabulary of commands used in other operating systems. It can suggest the appropriate UNIX command to a user who may have typed a command learned on another system.

This tutorial is also implemented in FRANZ LISP and runs under Berkeley 4.1a UNIX. It uses a knowledge representation system called "Frail" to represent it database. As far as performance, it is said to be reasonable, but like any other large LISP application on a timesharing machine, the overhead involved poses problems.

6.3 Comparisons

The C-PROLOG TUTOR uses preprogrammed sequences to create each scenario, while the LISP TUTOR creates instructional interactions depending on the progress of its user.

Unlike the other two, the Prolog system does not watch every character the user types. It does not provide hints or suggestions before an answer is input, but it does reword questions, give an explanation for each wrong answer, and always explains the correct answer. Explanations are purposely worded so as not to give away the entire answer before a retry.

This tutorial is much smaller in size than the LISP TUTOR, and judging from the capabilities of CLYDE, it, too, is probably larger. Each tool covers a wide variety of topics about the particular subject area being taught and gives the user the opportunity to learn each topic by doing.

The C-PROLOG TUTOR allows the user to leave the tutorial in a variety of ways. If someone is used to ending a program by typing "exit", "quit", or "q", TUTOR interprets it as a "halt", which is the correct way to leave PROLOG. This feature was designed purposely, in the same way CLYDE recognizes commands from other systems, so the user would have one less obstacle to overcome.

7. Conclusions

Overall, the development of TUTOR proved to be a very fruitful and worthwhile project. It was beneficial not only from the viewpoint of learning the Prolog language better, but the experience and knowledge necessary to accomplish such a task was gained.

This particular application was not well-suited for Prolog, but in doing it, many features of the language, such as backtracking and the use of its database, were learned and often utilized. It took time to learn how to think in terms of how Prolog works, which is not like any other algorithmic language. Prolog is definitely better used in applications like expert systems and relational databases where facts, rules and questions are constantly needed. Most of the goals in TUTOR simply print text to the screen.

Each of the major requirements established in the beginning of this project was implemented. The system operates as intended and contains a complete set of material for someone to learn basic Prolog. Every decision and problem encountered during this project has been described. TUTOR was written with simplicity and structure in mind. Its operation is well documented. The code is composed of many small files which should make additions fairly simple to implement.

8. Extensions

There are many other areas of Prolog that are not taught by this tutorial. Any of those could be added to this system. Described here are some of the more important and useful areas that could be considered.

One area of PROLOG never mentioned in this project is the debugger. Throughout this entire effort, the debugger was in constant use. To learn how a predicate works, to see which goals are resatisfied by backtracking, to watch where a rule was failing, and to examine the contents of different variables, are all prime examples of how it was used. The Prolog debugger is a very useful mechanism that is a must for developing a program of any reasonable size. Without it, this package would have been very difficult to develop.

Improvements could be added to benefit the experienced user. Perhaps a wider range of topics, only accessible by that level, would attract someone who already has a Prolog background. The option to skip a summary or exercise may also be attractive. Exercises that give the user more chances to actually program would help someone experienced as well as users in the other levels.

Bibliography

[1] Anderson, John R. and Reiser, Brian J., "The LISP TUTOR", Byte, April, 1985.

[2] Bailey, Robert W., "Human Performance Engineering: a Guide for System Designers", Prentice-Hall, 1982.

[3] Barr, Avron and Feigenbaum, Edward, "The Handbook of Artificial Intelligence", Department of Computer Science, Stanford University, William Kaufmann, Inc., 1982.

[4] Clark, K.L. and Tarnlund, S.A., "Logic Programming", Academic Press, 1982.

[5] Clocksin, W.F. and Mellish, C.S.,"Programming in Prolog", Springer-Verleg, Berlin Heidelberg, 1981.

[6] Dean, M., "How a computer should talk to people.", IBM Systems Journal, Vol. 21, No. 4, 1982.

[7] Hebditch, D., "Dialogue Design for user-friendly systems", User-Friendly Systems, Infotech State of the Art Report, Series 9, Number 4, Pergamon Infotech Limited, Berkshire, England, 1981.

[8] Hogger, Christopher John, "Introduction to Logic Programming", Academic Press, 1984.

[9] IEEE Computer Society, "The First Conference on Artificial Intelligence Applications", IEEE Computer Society Press, Los Angeles, CA., 1984.

[10] Irgon, Adam E. and Martin, John C., "Clyde: A UNIX Tutor", Human-Computer Interaction, Elsevier Science Publishers, Amsterdam, 1984.

[11] Kowalski, R., "Logic as the Fifth Generation Computer Language", The Fifth Generation Computer Project, State of the Art Report, 11:1, Pergamon Infotech Limited, 1983.

[12] Meredith, J. C., "The CAI Author/Instructor", Educational Technology Publications, Englewood Cliffs, NJ, 1971.

[13] Pereira, Fernando, editor, "C-Prolog User's Manual", SRI International, Menlo Park, California, 1984.

[14] Rubinstein, Richard, and Hersh, Harry, M., "The Human Factor - Designing Computer Systems for People", Digital Press, 1984.

[15] Sleeman,D. and Brown,J.S., "Intelligent Tutoring Systems", Academic Press, New York, 1982.

[16] Warren, D.H.D., "A View of the Fifth Generation and its Impact", The Fifth Generation Computer Project, State of the Art Report, 11:1, Pergamon Infotech Limited, 1983.

APPENDIX 1

TUTOR manual page

NAME
    TUTOR - a tutorial in C-Prolog

SYNOPSIS
```
        $ prolog  (RETURN)
        ----------------
        C-Prolog version 1.4

        yes
        | ?- ['/usr/prolog/tutor'].  (RETURN)
             -----------------------------------
```

DESCRIPTION
    This manual page describes how to use a Prolog tutorial called TUTOR. TUTOR is an interactive, instructional tool that teaches a version of Prolog called C-Prolog. It is written in C-Prolog and is available on the VAX 11/780, Perkin Elmer 8/32 and the PLEXUS systems at Kansas State University.

GETTING STARTED
    TUTOR accommodates users of three skill levels. The history of Prolog and an overview of C-Prolog are described in the first two lessons. The major areas of the language are covered in the next sixteen lesson/summary/exercise sections. The last lesson briefly describes the core set of builtin predicates of C-Prolog.

TUTOR is supported on the following five machines at Kansas State.

VAX 11/780	(KSUVAX1)
Perkin Elmer 8/32	(KSU832)
Perkin Elmer 3220	(KSU3220)
Plexus 1	(KSUPLX1)
Plexus 2	(KSUPLX2)

Each of these machines has Version 1.4 of the C-Prolog interpreter available. In all cases, to invoke the tutorial, you must first get into the interpreter by typing

```
        $ prolog  (RETURN)
        ----------------
```

The convention (RETURN) means you should type a carriage return. It is used throughout this manual page and the tutorial. When TUTOR is loaded, you will see the following response and prompt.

```
C-Prolog version 1.4

| ?-
```

At this time, C-Prolog is ready for user input. If you have problems with the tutorial from this point on, consult the "HAVING PROBLEMS" section. It describes various symptoms and what to do if you see them.

To load the tutorial, the file "tutor" must be consulted (read in). After the prompt shown above, type the following

```
| ?- ['/usr/prolog/tutor']. (RETURN)
     ---------------------------------
```

By consulting this file, all the other files necessary to start up TUTOR will be consulted automatically. As each file is read into the database, its size in bytes is printed along with the amount of time it took to be read in (these times may vary, depending on the machine). Instructions are also given to start up the tutorial or leave the interpreter. You can expect the following output.

```
/usr/prolog/lib consulted 3456 bytes 1.03333 sec.
/usr/prolog/start consulted 1184 bytes 0.4 sec.
/usr/prolog/ask consulted 1160 bytes 0.233335 sec.
/usr/prolog/mult consulted 2216 bytes 0.533335 sec.
/usr/prolog/repeatt consulted 1676 bytes 0.633334 sec.
/usr/prolog/get_it consulted 2904 bytes 1.11667 sec.

************************************************
***    To begin, type   start. (RETURN)     ***
***                     ---------------     ***
***                                         ***
***    To quit,  type   halt. (RETURN)      ***
***                     --------------      ***
************************************************

/usr/prolog/tutor consulted 12596 bytes 4.3 sec.
yes
| ?-
```

RUNNING TUTOR

Now that TUTOR is loaded, you will see the prompt "?-", which tells you that Prolog is ready for a command. At this point, follow the instructions to either start up or leave the tutorial.

REMEMBER THIS:
Prolog must see a period after every answer.
But, don't worry if you type the answer and

then hit a RETURN without the period. You can type the period at the next prompt followed by a RETURN and Prolog won't know the difference.

If you type in a variable at this time (a word beginning with a capital letter or an underscore), Prolog will respond with "Statement is a variable!" and then the "| ?-" prompt again.

When TUTOR is started, a welcome screen will appear. It describes each of the skill levels available for using the tool.

```
     Beginner      - Assumes no prior knowledge
                     of PROLOG
                   - Automatically guides you through
                     every topic
     Intermediate  - Allows you to resume from where
                     you left off during last session,
                     or to choose one topic at a time
     Experienced   - Allows you to pick one particular
                     topic at a time
```

A beginner will sequentially be taken through each of the 19 Prolog topics listed below. Intermediate users can choose to continue from a certain topic or like the experienced user, can select one topic at a time.

```
1) history of PROLOG        7) rules  13) lists
2) C-PROLOG at KSU          8) syntax 14) backtracking
3) facts             9) characters     15) cut
4) questions               10) operators
5) variables               11) equality
6) conjunctions               12) arithmetic

16) reading/writing characters  18) accessing files
17) reading/writing terms        19) built-in predicates
```

Each item is covered in a seperate lesson. Topics 3 through 18 also include a summary and exercise.

TO REPEAT A TOPIC

Many lessons revert back to previous topics. TUTOR allows the beginner or beginner/intermediate user to go back and review a topic covered in an earlier lesson. If you see a topic you want to review, wait until the end of the lesson, summary or exercise you are on and the repeat command will appear as one of the choices across the bottom of the screen.

```
TO REPEAT A TOPIC - type "r."
```

When you type "r.", TUTOR tells you it will take you back to where you started after the topic is repeated. A menu of topics will appear on the screen, but it will contain only those topics that precede the topic you are currently working on. If you are working on Lesson 5, for instance, this is the dialog and menu you would see.

```
After the topic you choose is repeated, TUTOR will
continue from where you left off.  These are the
topics preceding the lesson you are currently working
on:

1) History of Prolog          2) C-PROLOG at KSU
3) facts            4) questions
```

You would then be asked,

```
Which topic do you want to repeat? (ie "2. (RETURN)")
```

If the topic you chose contains a lesson, summary and exercises, that is what you'll see. When the entire topic has been repeated, TUTOR starts up the next section of the tutorial immediately.

LEAVING TUTOR

There are a number of ways to leave this tutorial. Whenever the message

```
TO QUIT - type "q."
```

appears on the screen, you have the option to leave TUTOR. Besides "q.", TUTOR will also recognize "quit." , "exit." and "halt." as termination commands. Before the session is over, you are first asked

```
Do you want to try a new skill level? ("y." / "n.")
```

in case the current level is too easy or too difficult. If another skill level is your choice, you will see this next question.

```
Which new level do you want?
     Beginner     -- type "b.  (RETURN)"
     Intermediate -- type "i.  (RETURN)"
     Experienced  -- type "e.  (RETURN)"
```

If not, an "n." will cause the following message to be printed, then take you completely out of Prolog and back to the UNIX prompt.

```
You have reached the end of the PROLOG tutorial.
Feel free to use it again anytime for a refresher
or to continue from where you left off.

[ Prolog execution halted ]
$
```

In the beginner mode or the intermediate mode where the continue option was chosen, TUTOR will stop automatically when it has reached the end of the topic sequence. At this time, you will also see the above message appear on the screen.

HAVING PROBLEMS ??

SYMPTOM	WHAT TO DO
no | ?-	Prolog did not understand your input. Try again.
The prompt "|" keeps repeating	Prolog is probably waiting for a period. Type "." (RETURN).
The prompt "|:" keeps repeating	Prolog is waiting for a response. Type in one of the commands at the bottom of the screen or a response to the last question, followed by a period.
|: Action(h for help)	Type "a" (RETURN) and you will see "[execution aborted]" and the prompt "| ?-" on the screen. To start up the tutorial again, type "start." and a (RETURN). If you were using the beginner level, you may want to start up in the intermediate level so you don't have to repeat previous topics.
$ (Unix prompt)	The tutorial and Prolog were exited. You must re-enter Prolog and reload "tutor" to use the tutorial.
TUTOR gets stuck somewhere. No input works.	Hit BREAK or DEL and follow directions on previous page for the symptom "|: Action (h for help)

FILES

/usr/prolog/*

SEE ALSO

Clocksin, W.F. and Mellish, C.S.,"Programming in Prolog", Springer-Verlag, Berlin Heidelberg, 1981.

Pereira, Fernando, editor, "C-Prolog User's Manual", SRI International, Menlo Park, California, 1984.

APPENDIX 2

TUTOR:User's Guide

TUTOR USER'S GUIDE

A Tutorial that teaches

C-PROLOG

on the

VAX 11/780,

Perkin Elmer 8/32

and

Plexus

Machines

Kansas State University
Computer Science Department

November 1, 1985

CONTENTS

1. Introduction

TUTOR evolved as an exercise in PROLOG; to study its characteristics, learn its structure and then apply it in the development of an interactive software tool. It was developed to provide an independent, self-paced means to learn the C-PROLOG version of PROLOG. It is a tutorial designed for both new and experienced users and contains a comprehensive set of lessons, summaries and exercises to explain each basic area of the language.

TUTOR is written in C-PROLOG. It presents C-PROLOG in the same structural manner used by Clocksin and Mellish in their book which describes standard PROLOG.

2. Getting Started

TUTOR is supported on the following five machines at Kansas State.

VAX 11/780	(KSUVAX1)
Perkin Elmer 8/32	(KSU832)
Perkin Elmer 3220	(KSU3220)
Plexus 1	(KSUPLX1)
Plexus 2	(KSUPLX2)

Each of these machines has Version 1.4 of the C-Prolog interpreter available.

In all cases, to invoke the tutorial, you must first get into the interpreter by typing

```
        $ prolog  {RETURN}
        ----------------
```

The convention {RETURN} means you should type a carriage return. It is used throughout this manual and the tutorial. When prolog is loaded, you will see the following response and prompt.

```
        C-Prolog version 1.4
        | ?-
```

At this time, C-Prolog is ready for user input. If ou have problems with the tutorial from this point on, consult section 6 of this manual entitled "Having Problems ?". It describes various symptoms and what to do if you see them.

To load the tutorial, the file "tutor" must be consulted. After the prompt shown above, type the following

```
        | ?-['/usr/prolog/tutor'].  {RETURN}
            --------------------------------
```

By consulting this file, all the other files necessary to start up TUTOR will be consulted automatically. As each file is read into the database, its size in bytes is printed along with the amount of time it took to be read in.

Instructions are also given to start up the tutorial or leave the interpreter. The following output can be expected, but the numbers don't have to be exact.

```
/usr/prolog/lib consulted 3456 bytes 1.03333 sec.
/usr/prolog/start consulted 1184 bytes 0.4 sec.
/usr/prolog/ask consulted 1160 bytes 0.233335 sec.
/usr/prolog/mult consulted 2216 bytes 0.533335 sec.
/usr/prolog/repeatt consulted 1676 bytes 0.633334 sec.
/usr/prolog/get_it consulted 2904 bytes 1.11667 sec.

************************************************
***     To begin, type   start. {RETURN}     ***
***                      ---------------     ***
***                                          ***
***     To quit,  type   halt. {RETURN}      ***
***                      --------------      ***
************************************************

/usr/prolog/tutor consulted 12596 bytes 4.3 sec.
yes
| ?-
```

3. **Running TUTOR**

Now that TUTOR is loaded, you will see the prompt "?-", which tells you that Prolog is ready for a command. At this point, follow the instructions to either start up or leave the tutorial.

REMEMBER THIS:

Prolog must see a period after every answer. But, don't worry if you type the answer and then hit a RETURN without the period. You can type the period at the next prompt followed by a RETURN and Prolog won't know the difference.

If you type in a variable at this time, a word beginning with a capital letter or an underscore, Prolog will respond with "Statement is a variable!" and then the "| ?-" prompt again.

When TUTOR is started, a welcome screen will appear. It describes each of the skill levels available for using the tool.

Beginner	- Assumes no prior knowledge of PROLOG - Automatically guides you through every topic
Intermediate	- Allows you to resume from where you left off during last session, or to choose one topic at a time
Experienced	- Allows you to pick one particular topic at a time

A beginner will sequentially be taken through each of the 19 Prolog topics listed below. Intermediate users can choose to continue from a certain topic or like the experienced user, can select one topic at a time.

```
1) history of PROLOG   7) rules          13) lists
2) C-PROLOG at KSU     8) syntax         14) backtracking
3) facts               9) characters     15) cut
4) questions          10) operators
5) variables          11) equality
6) conjunctions       12) arithmetic

16) reading/writing characters  18) accessing files
17) reading/writing terms       19) built-in predicates
```

Each item is covered in a separate lesson. Topics 3 through

18 also include a summary and exercise.

4. To Repeat a Topic

Many lessons refer back to previous topics. TUTOR allows the beginner or beginner/intermediate user to go back and review a topic covered in an earlier lesson. If you see a topic you want to review, wait until the end of the current lesson, summary or exercise and the repeat command will appear as one of the choices across the bottom of the screen.

TO REPEAT A TOPIC - type "r."

When you type "r.", TUTOR tells you it will take you back to where you started after the topic is repeated. A menu of topics will appear on the screen, but it will contain only those topics that precede the topic you are currently working on. If you are working on Lesson 5, for instance, this is the dialog and menu you would see.

```
After the topic you choose is repeated, TUTOR will
continue from where you left off.  These are the
topics preceding the lesson you are currently working
on:

1) History of Prolog                 2) C-PROLOG at KSU
3) facts                             4) questions
```

You would then be asked,

```
        Which topic do you want to repeat? (ie. "2. (RETURN)")
```

If the topic you chose contains a lesson, summary and exercise, that is what you'll see. When the entire topic has been repeated, TUTOR starts up the next section of the tutorial immediately.

5. Leaving TUTOR

There are a number of ways to leave this tutorial. Whenever the message

```
        TO QUIT - type "q."
```

appears on the screen, you have the option to leave TUTOR. Besides "q.", TUTOR will also recognize "quit." , "exit." and "halt." as termination commands. Before the session is over, you are first asked

```
        Do you want to try a new skill level? ("y." / "n.")
```

in case the current level is too easy or too difficult. If another skill level is your choice, you will see this next question.

```
        Which new level do you want?
                Beginner     -- type "b.  (RETURN)"
                Intermediate -- type "i.  (RETURN)"
                Experienced  -- type "e.  (RETURN)"
```

If not, an "n." will cause the following message to be printed, then take you completely out of Prolog and back to the UNIX prompt.

```
You have reached the end of the PROLOG tutorial.
Feel free to use it again anytime for a refresher
or to continue from where you left off.

[ Prolog execution halted ]
$
```

In the beginner mode or the intermediate mode where the continue option was chosen, TUTOR will stop automatically when it has reached the end of the topic sequence. At this time, you will also see the above message appear on the screen.

6. Having Problems ??

SYMPTOM	WHAT TO DO
no | ?-	Prolog did not understand your input. Try again.
The prompt " keeps repeating	Prolog is probably waiting for a period. Type "." {RETURN}.
The prompt "|:" keeps repeating	Prolog is waiting for a response. Type in one of the commands at the bottom of the screen or a response to the last question, followed by a period.
|: Action(h for help)	Type "a" {RETURN} and you will see "[execution aborted]" and the prompt "| ?-" on the screen. To start up the tutorial again, type "start." and a {RETURN}. If you were using the beginner level, you may want to start up in the intermediate level so you don't have to repeat previous topics.
$ (Unix prompt)	The tutorial and Prolog were exited. You must re-enter Prolog and reload "tutor" to use the tutorial.
TUTOR gets stuck somewhere. No input works.	Hit BREAK or DEL and follow directions above for the symptom "|: Action (h for help)

7. Glossary of Terms

- arity - the number of arguments associated with a particular predicate. For example, in the fact "dances(jane,fast).", dances has an arity of two.

- builtin - a builtin predicate is one that has been previously defined by Prolog. It is a available to every Prolog user.

- fail - a goal will fail if it does not match at least one clause in the database.

- Fifth-generation - computer systems for the 1990's. Knowledge information processing done while incorporating new technologies of VLSI architecture, parallel processing, logic programming, knowledge based systems, artificial intelligence and pattern processing.

- functor - another name for a predicate. It defines the relationship between arguments.

- instantiated - a variable becomes instantiated when it assumes a value.

- interpreter - a software package that translates a program to executable form by translating and executing each line in turn without waiting to translate the program as a whole.

- placemarker - a mechanism used to mark the spot in the database of where the last match for a given goal occurs. A separate placemarker is kept for each goal.

- resatisfy - to resatisfy a goal, backtracking must occur. Every variable of the goal becomes uninstantiated. Beginning at the placemarker, the database is searched for another match. If a match is found, the goal is resatisfied.

- succeeds - a goal succeeds if it matches a clause in the database.

- unification - the process of substituting a value in place of a variable.

- uninstantiated - a variable is uninstantiated whenever it has no value.

- Von Neumann architecture - an architecture where programs are stored in the computer along with the data. Instructions could now be changed without rewiring hardware. Since instructions are stored as numbers, they could be processed as data.

APPENDIX 3

Source Code

```
/* Lesson 18 - ACCESSING FILES */

accfile:- newpg,
        write('Lesson 18 - ACCESSING FILES'),nl,nl,nl,
        write('Normally, the current input and current output stream in PROLOG is your'),nl,nl,
        write('terminal.  These are handled by a built-in file called "user.". If instead,'),nl,nl,
        write('you want to read data from or write data to another file, you can use '),nl,nl,
        write('the PROLOG predicates "see" and "tell".'),nl,nl,
        write('The goal "see(A)" switches the current input stream from the terminal to'),nl,nl,
        write('file "A", if A was instantiated to an atom which specified the filename.'),nl,nl,
        write('It opens the file for input and points to the top of the file.  The contents'),nl,nl,
        write('of the file can now be read using the "get", "get0" and "read" predicates.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('ACCESSING FILES - continued'),nl,nl,nl,
        write('When you want to close the file and switch back to standard terminal input,'),nl,nl,
        write('use the goal "seen", with no arguments.'),nl,nl,
        write('To find out what the current input stream is set to, the goal "seeing(B)"'),nl,nl,
        write('will instantiate B to that name, providing B was not already set to'),nl,nl,
        write('something else. '),nl,nl,nl,
        write('To change the current output stream to file D, you need the goal "tell(D)".'),nl,nl,
        write('It will open the file and direct all subsequent writing by predicates like '),nl,nl,
        write('"put" and "write", into the file until it is closed.  If file D already'),nl,nl,
        write('exists, it will be overwritten.  Otherwise a new file will be created.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('ACCESSING FILES - continued'),nl,nl,nl,
        write('The predicate "told" will close the file and switch the current output'),nl,nl,
        write('stream back to "user".  Like "seeing", there is a predicate called "telling"'),nl,nl,
        write('that will instantiate its argument to the name of the current output stream.'),nl,nl,nl,
        write('Both "see" and "tell" can only be satisfied once.  Backtracking over them'),nl,nl,
        write('will not change the current input or output stream back to its previous value.'),nl,nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('ACCESSING FILES - continued'),nl,nl,nl,
        write('Until now, we have discussed methods for reading data from a file into'),nl,nl,
        write('a program and also writing data from a program into a file.  A much more'),nl,nl,
        write('common use of files in PROLOG is to store programs.  All of the facts and rules'),nl,nl,
        write('that make up a program can become part of the database with one command,'),nl,nl,
        write('"consult".  C-PROLOG uses the square bracket notation or the consult predicate'),nl,nl,
        write('read in files.  The goal "['), put(39),
        write('/usr/prolog/tutor'), put(39),
        write(']." that you typed at the'),nl,nl,
        write('beginning of this session, brought in all the files needed to start up the'),nl,nl,
        write('tutorial.  To consult multiple files, a goal like   "[main, sub1, sub2, sub3]"'),nl,nl,
        write('would read the contents of all four files into the database.'),nl,nl,nl,
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Lesson 12 - ARITHMETIC */

arith:- newpg,
        write('Lesson 12 - ARITHMETIC'),nl,nl,nl,
        write('Although PROLOG is not the type of language heavily used for mathematics,'),nl,nl,
        write('it does have operators for doing comparisons and calculations.  There are'),nl,nl,
        write('six built-in predicates, in the form of infix operators that allow you'),nl,nl,
        write('to compare numbers.  They are contained in the following examples.'),nl,nl,nl,
        write('         A = B.           "A equals B"'),nl,
        write('         A \= B.          "A is not equal to B"'),nl,
        write('         A < B.           "A is less than B"'),nl,
        write('         A > B.           "A is greater than B"'),nl,
        write('         A =< B.          "A is less than or equal to B"'),nl,
        write('         A >= B.          "A is greater than or equal to B"'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('ARITHMETIC - continued'),nl,nl,
        write('Assume the database contains these facts and a rule.'),nl,nl,
        write(' born(frank,1925).'),nl,
        write(' born(william,1938).'),nl,
        write(' born(judy,1932).'),nl,nl,
        write(' child(X,Y,Z):- born(X,A),'),nl,
        write('              A > Y,'),nl,
        write('              A < Z.'),nl,nl,
        write('A conversation with PROLOG might then proceed as follows:'),nl,nl,
        write('         ?- child(X,1920,1935).  (RETURN)'),nl,
        write('            ----------------------------'),nl,
        write('         X = frank ;  (RETURN)'),nl,
        write('                  -----------'),nl,
        write('         X = judy ; (RETURN)'),nl,
        write('                 ----------'),nl,
        write('         no '),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('ARITHMETIC - continued'),nl,nl,nl,
        write('Operators like  "+", "-", "*"  and  "/" are very common but are not evaluated'),nl,nl,
        write('in PROLOG, unless the "is" operator is part of the same goal.  The "is"'),nl,nl,
        write('operator is infix like the other operators above.  It expects an unknown'),nl,nl,
        write('variable on its left and an arithmetic expression on its right.  It will'),nl,nl,
        write('evaluate the expression and instantiate the left side to the result.'),nl,nl,
        write('Some examples:'),nl,nl,
        write('         Pay is (hours*4.50)'),nl,nl,
        write('         X is Y/2 '),nl,nl,
        write('         Abc is (3*m+2/p+1)'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('ARITHMETIC - continued'),nl,nl,nl,
        write('The divide operator "/", denotes integer division.  An expression'),nl,nl,
        write('containing this operator will only return the integer portion of the'),nl,nl,
        write('quotient.'),nl,nl,
        write('         "Y is 9/2."          instantiates Y to 4 '),nl,nl,
        write('         "Dozen is 144/12."  instantiates Dozen to 12 '),nl,nl,
        write('To produce a remainder, the operator "mod", short for modulo, should'),nl,nl,
        write('be used.  ONLY the remainder is returned.'),nl,nl,
        write('         "Z is 9 mod 2."      instantiates Z to 1 '),nl,nl,
        write('         "W is 3 mod 1."      instantiates W to 0 '),nl,nl,nl,
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Routine that asks the user a question, interprets his reply,
   allows a retry and if necessary, explains the correct answer. */

ask(Q1, Q2, Ans, Reas):-  asserta(q2(Q2)),
                          asserta(ans(Ans)),
                          asserta(reas(Reas)),
                          a_out(Q1,1),
                          retract(reas(Reas)),
                          retract(ans(Ans)),
                          retract(q2(Q2)).

a_out(Q,N):- nl,nl,write(Q),nl,nl,
             next1,
             mid_input(R1),
             check(R1,N).


check(X,_):- ans(X),
             write(X),
             write(' is the correct answer.'),nl,nl,
             reas(R),
             write(R),nl,nl,nl,nl.

check(X,0):- write(X),
             write(' is incorrect.'),nl,nl,
             reas(R),
             write(R),nl,nl,nl,nl.

check(X,Y):- Z is Y-1,
             write(X),
             write(' is incorrect.'),nl,nl,nl,
             q2(Q3),
             a_out(Q3,Z).
```

```
/* Lesson 14 - BACKTRACKING */

backtr:- newpg,
        write('Lesson 14 - BACKTRACKING'),nl,nl,nl,
        write('When attempting to satisfy a goal, two things can happen.  If a match'),nl,nl,
        write('is found, the database is marked, variables are instantiated, and PROLOG'),nl,nl,
        write('moves to the next goal on the right.  If no match is found, that particular'),nl,nl,
        write('goal fails, and PROLOG "goes back" to the previous goal and tries to'),nl,nl,
        write('resatisfy it. '),nl,nl,nl,
        write('This is the whole idea behind BACKTRACKING and it is all done'),nl,nl,
        write('automatically by PROLOG.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('BACKTRACKING - continued'),nl,nl,nl,
        write('Assume the following facts are in the database.'),nl,nl,
        write('                    person(peter).'),nl,
        write('                    person(paul).'),nl,
        write('                    person(mary).'),nl,nl,
        write('The question "?-person(X)." would produce  X = peter ;  {RETURN}'),nl,
        write('                                                     ----------'),nl,
        write('                                           X = paul ;  {RETURN}'),nl,
        write('                                                    ----------'),nl,
        write('                                           X = mary ; {RETURN}'),nl,
        write('                                                    ---------'),nl,
        write('                                           no'),nl,nl,
        write('if we told it to resatisfy the goal, or BACKTRACK, after each answer was'),nl,nl,
        write('printed.  We force BACKTRACKING here with the ";" symbol.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('BACKTRACKING - continued'),nl,nl,nl,
        write('Each time BACKTRACKING occurs, PROLOG will undo what was accomplished by the'),nl,nl,
        write('previous goal.  Any variables, in that goal, that were instantiated lose their'),nl,nl,
        write('value.  A separate place-marker is kept for each goal that the database is'),nl,nl,
        write('searched for.  PROLOG will begin at the marker, not at the top of the'),nl,nl,
        write('database, each time it tries to resatisfy the goal.  If the goal is'),nl,nl,
        write('satisfied again, the place-marker is moved.'),nl,nl,nl,nl,nl,nl,nl,
        (c_level(s) -> next2; next3),
        end_input.
```

```
/* Lesson 19 - BUILTIN PREDICATES */

builtin:- newpg,
        write('Lesson 19 - BUILTIN PREDICATES'),nl,nl,nl,
        write('There are many functions already defined by PROLOG called builtin predicates.'),nl,nl,
        write('This lesson will give you a basic understanding of those in the "core" set.'),nl,nl,
        write('It does not explain how they work, but what they do and what they are used'),nl,nl,
        write('for.  Some of these may look familiar from earlier lessons.'),nl,nl,
        write('        atom()          integer()         reconsult()'),nl,
        write('        atomic()        member()          tab()'),nl,
        write('        consult()       nl                true'),nl,
        write('        fail            nonvar()          var()'),nl,nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('BUILTIN PREDICATES - continued'),nl,nl,
        write('atom(A)'),nl,
        write('======='),nl,
        write('This goal succeeds if A is a Prolog atom.  Remember, an atom consists of either'),nl,nl,
        write('    - lower case letters and digits and begins with a lower case letter'),nl,
        write('    - all symbols or '),nl,
        write('    - a combination of letters (upper and lower case), digits, and symbols'),nl,
        write('      all enclosed in single quotes.'),nl,nl,
        write('Here are some examples:'),nl,nl,
        write('        ?- atom(kitten).  (RETURN)'),nl,
        write('           -----------------------'),nl,
        write('        yes'),nl,
        write('        ?- atom('), put(39), write('Kits'), put(39), write(').  (RETURN)'),nl,
        write('           -----------------------'),nl,
        write('        yes'),nl,
        write('        ?- atom("hello").  (RETURN)'),nl,
        write('           -----------------------'),nl,
        write('        no'),nl,
        next2,
        mid_input(_),
        newpg,
        write('BUILTIN PREDICATES - continued'),nl,nl,nl,
        write('atomic(M)'),nl,
        write('========='),nl,nl,
        write('Very similar to "atom", the "atomic" goal tests its argument for either an'),nl,nl,
        write('integer or an atom.  Any argument that succeeds with "atom", also succeeds'),nl,nl,
        write('with "atomic".  For instance,'),nl,nl,nl,
        write('        ?- atom(1234).  (RETURN)         will fail'),nl,
        write('           --------------------'),nl,
        write('        no'),nl,
        write('        ?- atomic(1234).  (RETURN)       will pass'),nl,
        write('           ----------------------'),nl,
        write('        yes'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('BUILTIN PREDICATES - continued'),nl,nl,nl,
        write('consult(file).'),nl,
        write('=============='),nl,nl,
        write('Prolog programs can be stored in files.  The files can then be read into'),nl,nl,
        write('the database using the "consult" goal.  It reads the file, while checking'),nl,nl,
        write('syntax character by character.  Then to start the program, you just have to'),nl,nl,
        write('invoke the appropriate clause.'),nl,nl,
        write('There is no limit to the number of files consulted by one "consult" fact.'),nl,nl,
        write('It is easier though, to consult one file that contains the rest of the'),nl,nl,
        write('"consult" goals for the files you want read in.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
write('BUILTIN PREDICATES - continued'),nl,nl,nl,
write('fail'),nl,
write('===='),nl,nl,
write('Whenever you want to cause backtracking within a rule, the "fail" goal should'),nl,nl,
write('be used.  It has no arguments and it will never succeed.  As you know, when'),nl,nl,
write('a goal in Prolog fails, it automatically tries to resatisfy the preceding goal.'),nl,nl,
write('A rule containing this goal may look like this:'),nl,nl,nl,
write('                interview(X):- citizen(X), '),nl,
write('                               experience(X), '),nl,
write('                               criminal(X), fail.'),nl,nl,nl,
next2,
mid_input(_),
newpg,
write('BUILTIN PREDICATES - continued'),nl,nl,nl,
write('integer(G)'),nl,
write('=========='),nl,nl,
write('If G is instantiated to a whole number, "integer(G)" will succeed.  This goal'),nl,nl,
write('is used by TUTOR to make sure the topic lesson numbers input by the users'),nl,nl,
write('are really integers.'),nl,nl,
write('       ?- integer(7.5).  {RETURN}                fails'),nl,
write('          -----------------------'),nl,
write('       no'),nl,
write('       ?- integer(1032).  {RETURN}               succeeds'),nl,
write('          ------------------------'),nl,
write('       yes'),nl,nl,nl,nl,
next2,
mid_input(_),
newpg,
write('BUILTIN PREDICATES - continued'),nl,nl,nl,
write('nl'),nl,
write('=='),nl,
write('This goal means "newline".  It moves the cursor to the beginning of the next'),nl,nl,
write('line.  If backtracking occurs, it cannot be resatisfied.  In other words,'),nl,nl,
write('another newline is not output.'),nl,nl,nl,
write('nonvar(P)'),nl,
write('========='),nl,
write('If P is instantiated, it is no longer a variable.  When the argument of'),nl,nl,
write('"nonvar" is not a variable, the goal will succeed.  This predicate can be'),nl,nl,
write('handy for checking user input before passing it to another clause or'),nl,nl,
write('asserting it into the database.'),nl,nl,
next2,
mid_input(_),
newpg,
write('BUILTIN PREDICATES - continued'),nl,nl,nl,
write('reconsult(fileb)'),nl,
write('================'),nl,
write('Like consult, "reconsult" reads the facts and rules from a file into the'),nl,nl,
write('Prolog database.  But, while doing this, if it finds any existing clauses'),nl,nl,
write('whose predicate matches the predicate it is reading, they are overwritten.'),nl,nl,
write('Its main use is to correct programming errors.'),nl,nl,nl,
write('tab(Q)'),nl,
write('======'),nl,
write('To output 10 spaces, you would use the goal "tab(10)".  Like "nl", it succeeds'),nl,nl,
write('only once.  It should not be confused with the tab key on the terminal or a'),nl,nl,
write('typewriter.  It will only print the number of spaces you tell it to.'),nl,nl,
next2,
mid_input(_),
newpg,
write('BUILTIN PREDICATES - continued'),nl,nl,nl,
write('true'),nl,
write('===='),nl,nl,
write('Opposite of "fail", this goal always succeeds.  It is not needed in most'),nl,nl,
write('cases, but exists for convenience.  The clause "charlie(male):- true." can'),nl,nl,
write('be written as the fact "charlie(male)."  The "true" goal is not necessary.'),nl,nl,nl,
write('var(L)'),nl,
write('======'),nl,nl,
write('When L is uninstantiated,  it is a variable.  With no value,'),nl,nl,
write('"?-var(L)."  will succeed and return "yes".'),nl,nl,nl,nl,
(c_level(a) -> next2; next3),
end_input.
```

```
/* Lesson 9 - CHARACTERS */

chars:- newpg,
        write('Lesson 9 - CHARACTERS'),nl,nl,nl,
        write('There are two types of CHARACTERS in PROLOG: printing and non-printing.'),nl,nl,
        write('These are all the available printing CHARACTERS:'),nl,nl,
        write('         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 '),nl,nl,
        write('         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 '),nl,nl,
        write('         0 1 2 3 4 5 6 7 8 9 '),nl,nl,
        write('         ! " # $ % & ( ) = - ~ ^ | \ } ] { [ _ ` + ; : < > , . ? / '),
        put(39), put(32), put(64),nl,nl,
        write('CHARACTERS are considered integers with values between 0 and 127.  These'),nl,nl,
        write('values are assigned to each character by a standard ASCII code.  Printing'),nl,nl,
        write('characters are greater than ASCII 32 and non-printing are from 0 to 32.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('CHARACTERS - continued'),nl,nl,nl,
        write('Common non-printing characters are:'),nl,nl,
        write('         CHARACTER                ASCII VALUE'),nl,
        write('         ---------                -----------'),nl,
        write('         space                        32     '),nl,
        write('         CTRL C (interrupt)            3     '),nl,
        write('         CTRL Z (end-of-line)         26     '),nl,nl,
        write('Some ASCII values for non-printing CHARACTERS are machine dependent. These are'),nl,nl,
        write('the printing characters, their ASCII values and associated meanings:'),nl,nl,
        write(' ASCII Code     CHARACTER              MEANING'),nl,
        write(' ----------     ---------              -------'),nl,
        write('    33              !           exclamation mark: the "cut" symbol'),nl,
        write('    34              "           double quote: delimits strings '),nl,
        write('    35              #           sharp sign'),nl,
        write('    36              $           dollar sign'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('CHARACTERS - continued'),nl,nl,nl,
        write(' ASCII Code     CHARACTER              MEANING'),nl,
        write(' ----------     ---------              -------'),nl,
        write('    37              %           percent sign'),nl,
        write('    38              &           ampersand'),nl,
        write('    39              '),
        put(39),
        write('         single quote: surrounds argument of write'),nl,
        write('    40              (           parenthesis: for grouping and structures'),nl,
        write('    41              )           closing parenthesis'),nl,
        write('    42              *           asterisk: "multiply" in "is" goals'),nl,
        write('    43              +           plus: "add" in "is" goals'),nl,
        write('    44              ,           comma: conjunction of goals and '),nl,
        write('                                       separates arguments '),nl,
        write('    45              -           minus: "subtract" in "is" goals'),nl,
        write('    46              .           period: ends clauses'),nl,
        write('    47              /           backslash: "division" in "is" clauses'),nl,
        write('    48-57          0-9          digits'),nl,
        write('    58              :           colon'),nl,
        write('    59              ;           semicolon: disjunction of goals'),nl,
        write('    60              <           less than: "less than" in "is" goals'),nl,nl,
        next2,
        mid_input(_),
        newpg,
write('CHARACTERS - continued'),nl,nl,
write(' ASCII Code      CHARACTER                 MEANING'),nl,
write(' ----------      ---------                 -------'),nl,
write('    61               =             equals: equality predicate'),nl,
write('    62               >             greater than: "greater than" in "is" goals'),nl,
write('    63               ?             question mark'),nl,
write('    64               '),
put(64),
write('          "at" sign'),nl,
write('    65-90            A-Z           upper-case A through Z'),nl,
write('    91               [             square bracket: starts a list'),nl,
write('    92               \             backslash'),nl,
write('    93               ]             closing square bracket: ends a list'),nl,
write('    94               ^             caret or up-arrow'),nl,
write('    95                             underscore: anonymous variable'),nl,nl,
write('    96               `             accent'),nl,
write('    97-122           a-z           lower-case a through z '),nl,
write('    123              {             curly bracket'),nl,
write('    124              |             bar: separates "head" and "tail" in  a list'),nl,
write('    125              }             closing curly bracket'),nl,
write('    126              ~             tilde'),nl,nl,
next2,
mid_input(_),
newpg,
write('CHARACTERS - continued'),nl,nl,nl,
write('You might ask,  why would you use an ASCII code?  When using the'),nl,nl,
write('"write" predicate to output a sentence, for instance, it expects its argument'),nl,nl,
write('surrounded by single quotes.  If you want to output a single quote as part'),nl,nl,
write('of the sentence, you cannot just type the quote mark because "write" will'),nl,nl,
write('think it sees the end of its argument.  To output a sentence like:'),nl,nl,
write('         "That'), put(39), write('s his new car."'),nl,nl,
write('you will need three goals. '),nl,nl,
write('         write('), put(39), write('That'), put(39),
write('), put(39), write('), put(39),
write('s his new car.'), put(39), write('),'),nl,nl,
write('to handle the apostrophe in the first word.'),nl,nl,nl,
(c_level(e) -> next2; next3),
end_input.
```

```
/* Lesson 6 - CONJUNCTIONS */

conj:- newpg,
       write('Lesson 6 - CONJUNCTIONS'),nl,nl,nl,
       write('In earlier lessons, a question was a goal about one relationship.  The'),nl,nl,
       write('question "Does Susan like swimming and running?" is more complicated in'),nl,nl,
       write('that it involves multiple relationships.  One way to answer it would be'),nl,nl,
       write('to ask PROLOG if Susan likes swimming and if a "yes" was returned, then'),nl,nl,
       write('ask if Susan likes running.  When both goals are satisfied, the answer'),nl,nl,
       write('to the whole question is "yes".'),nl,nl,nl,
       write('A CONJUNCTION in PROLOG combines multiple goals, separated by commas,'),nl,nl,
       write('into one question.'),nl,nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('CONJUNCTIONS - continued'),nl,nl,nl,
       write('"Does Susan like swimming and running?" is written like:'),nl,nl,
       write('     ?- likes(susan,swimming), likes(susan,running).'),nl,nl,nl,
       write('The comma is pronounced "and".  When PROLOG sees a CONJUNCTION it attempts'),nl,nl,
       write('to satisfy one goal at a time, from left to right.  If every goal is'),nl,nl,
       write('satisfied, PROLOG returns a "yes".  As soon as one fails, PROLOG goes back'),nl,nl,
       write('to the previous goal and tries to resatisfy it.  In this particular case,'),nl,nl,
       write('the whole CONJUNCTION would fail because it does not contain any variables'),nl,nl,
       write('to resatisfy. Only one answer is given by PROLOG for each conjunction.  It'),nl,nl,
       write('does not produce an external answer for each goal.'),nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('CONJUNCTIONS - continued'),nl,nl,nl,
       write('Examples of CONJUNCTIONS:'),nl,nl,
       write('"Will Harry eat fish and chips?" translates to'),nl,nl,
       write('  ?- eat(harry,fish), eat(harry,chips).  '),nl,nl,
       write('"Do Judy and Mark love each other?" translates to'),nl,nl,
       write('  ?- love(judy,mark), love(mark,judy).   '),nl,nl,nl,
       write('CONJUNCTIONS like these contain only constants and thus are much less'),nl,nl,
       write('work for PROLOG.  To ask "Do Thomas and Rosemary sing the same kind of'),nl,nl,
       write('music?", a variable is involved.  We do not know what kind of music either'),nl,nl,
       write('person sings, if any at all.'),nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('CONJUNCTIONS - continued'),nl,nl,nl,
       write('The CONJUNCTION  "?- sings(thomas,X), sings(rosemary,X)."  is really'),nl,nl,
       write('asking two things.  Does Thomas sing X type of music AND does Rosemary'),nl,nl,
       write('also sing X type of music?'),nl,nl,
       write('PROLOG tries to find a match for the first goal.  Lets say it found'),nl,nl,
       write('"sings(thomas,blues)." in the database.  It would mark that place in the'),nl,nl,
       write('database and X would be instantiated to "blues".  Because the second'),nl,nl,
       write('goal also contains an X, it is also instantiated to "blues".  Now the'),nl,nl,
       write('second goal, "sings(rosemary,blues)." must be found.  Since it is a'),nl,nl,
       write('different goal, PROLOG begins searching from the top of the database.'),nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('CONJUNCTIONS - continued'),nl,nl,nl,
       write('If a match is found, that place is also marked and the CONJUNCTION is'),nl,nl,
       write('solved.  Other solutions may exist.  They are found the same way another'),nl,nl,
       write('answer to a question is found; entering a semi-colon and a RETURN after'),nl,nl,
       write('the answer is given.  If "sings(rosemary,blues)." was not found, PROLOG'),nl,nl,
       write('would automatically go back and try to re-satisfy the first goal.  The'),nl,nl,
       write('search would begin at the place that was marked for that particular goal.'),nl,nl,
       write('Whenever PROLOG has to re-satisfy a goal, it is called BACKTRACKING.'),nl,nl,
       write('That topic will be covered in a later lesson.'),nl,nl,nl,nl,
       (c_level(e) -> next2; next3),
       end_input.
```

```
/* Lesson 2 - C-Prolog information */

cpro:- newpg,
       write('Lesson 2 -  C-PROLOG at Kansas State'),nl,nl,nl,
       write('C-PROLOG is a version of PROLOG written in C language for 32 bit machines.'),nl,nl,
       write('It was developed at the University of Edinburgh in Scotland.  The system'),nl,nl,
       write('consists of a PROLOG interpreter and a number of built in, or system defined'),nl,nl,
       write('procedures.  It offers an interactive programming environment with tools to'),nl,nl,
       write('incrementally build programs, debug them by following their executions, and'),nl,nl,
       write('modify parts of them without having to start over from scratch.'),nl,nl,nl,
       write('The text of a program is normally contained in a number of files created by'),nl,nl,
       write('a text editor.  Each of these files can then be read in or "consulted" '),nl,nl,
       write('by C- OLOG.'),nl,
       next2,
       mid_input(_),
       newpg,
       write('C-PROLOG at Kansas State - continued'),nl,nl,nl,
       write('When you typed "['), put(39),
       write('/usr/prolog/tutor'), put(39),
       write(']." at the beginning of this tutorial, you'),nl,nl,
       write('told C-PROLOG to consult the file, tutor, which essentially read in all the'),nl,nl,
       write('files necessary to start up this tutorial.  Each time a file is consulted, the'),nl,nl,
       write('interpreter prints the number of bytes in that file and the time it took to'),nl,nl,
       write('read it in.  As you proceed, you will notice various other files being '),nl,nl,
       write('consulted as they are needed.'),nl,nl,nl,
       write('PROLOG and TUTOR are available on the VAX 11/780, Perkin Elmer 8/32 '),nl,nl,
       write('and on the PLEXUS machines on campus.'),nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('C-PROLOG at Kansas State - continued'),nl,nl,nl,
       write('One of the best textbooks available on PROLOG is called "Programming'),nl,nl,
       write('in PROLOG" by Clocksin and Mellish.  Its basic structure was used as'),nl,nl,
       write('as a model in the development of this tutorial.  The "C-PROLOG Users Manual"'),nl,nl,
       write('is also available on campus.'),nl,nl,nl,nl,nl,nl,nl,
       (c_level(e) -> next2; next3),
       end_input.
```

```
/* Lesson 15 - CUT */

cutt:- newpg,
       write('Lesson 15 - CUT'),nl,nl,nl,
       write('There is a special function in PROLOG called CUT, which is used to control'),nl,nl,
       write('backtracking.  CUT itself is a goal.  Its predicate is "!", the exclamation'),nl,nl,
       write('point, and it has no arguments.  It always succeeds once, but can never'),nl,nl,
       write('be resatisfied.  Backtracking over a CUT always fails and therefore prevents'),nl,nl,
       write('any goal before it from ever being resatisfied.'),nl,nl,
       write('It is a mechanism used to tell PROLOG that any decisions made, within that'),nl,nl,
       write('rule, before the CUT, cannot be changed.  When PROLOG sees a rule that contains'),nl,nl,
       write('a CUT, it knows that for all goals up to the CUT, including the goal in the'),nl,nl,
       write('head of the rule, it should not save a place-marker in the database.'),nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('CUT - continued'),nl,nl,nl,
       write('Suppose you are going to buy a new car.  You know exactly what kind you'),nl,nl,
       write('want and which options you cannot do without.  The color does not matter.'),nl,nl,
       write('If you walk into a Nissan showroom and say,'),nl,nl,
       write('  "I want to buy the FIRST Maxima you can find that is an automatic,'),nl,
       write('   with a sunroof and does not have leather seats.  The color and'),nl,
       write('   other options are not important.  Call me as soon as you find it."'),nl,nl,
       write('A rule like this will find your car if it is available.'),nl,nl,
       write('             car(X,Y,Z):- model(X,maxima),'),nl,
       write('                          transmission(X,automatic),'),nl,
       write('                          sunroof(X,yes),'),nl,
       write('                          leather_seats(X,no),!,'),nl,
       write('                          color(X,Y),'),nl,
       write('                          roof_rack(X,Z).'),nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('CUT - continued'),nl,nl,nl,
       write('The previous example showed CUT being used to produce one answer and'),nl,nl,
       write('prevent PROLOG from searching for alternative solutions.'),nl,nl,nl,nl,
       write('CUT is also commonly used in conjunction with the predicate "fail".'),nl,nl,
       write('There might be times when you want PROLOG to fail a particular goal'),nl,nl,
       write('without looking for other solutions.  The sequence " !, ail " will'),nl,nl,
       write('accomplish this because the goal "fail" will always fail and the CUT'),nl,nl,
       write('will prevent further backtracking from occurring.'),nl,nl,nl,nl,
       (c_level(e) -> next2; next3),
       end_input.
```

```
/* Exercise 18 - ACCESSING FILES */

e_accfile:-newpg,
        write('Exercise 18 - ACCESSING FILES'),nl,nl,nl,

        ask('What file does Prolog use for terminal input and output ?',
            'Terminal I/O uses a specific filename in Prolog.  What is it ?',
            user, 'The file "user" is another name for terminal I/O .'),

        ask('To begin reading from a file, which predicate is used ?',
            'What predicate will let you start reading from a file ?',
            see, 'The goal "see(filea)" will change the input stream from "user" to "filea".'),

        mult('Which goal will tell you the current input stream ?',
             'seen',
             'seeing(X)',
             'put(Y)',
             'No, this predicate closes the current input file.',
             'This goal will instantiate X to the name of the current input file.',
             'No, "put" will send a character to the current output stream.', b),
        next2,
        mid_input(_),

        mult('When would "tell(XYZ)" create AND open a file for output ?',
             'If "XYZ" was instantiated to a filename that did not already exist.',
             'always',
             'If "XYZ" was uninstantiated',
             'If the file already existed, it would be overwritten.',
             'No, a new file would is not always created by the "tell" predicate.',
             'No, if "XYZ" was uninstantiated, the goal "tell(XYZ)" would fail.', a),
        next2,
        mid_input(_),


        ask('How any arguments do "seen" and "told" each have ? (enter a number)',
            'The predicates "seen" and "told" require how many arguments ? (enter a number)',
            0, 'Both "seen" and "told" have zero arguments.'),


        mult('Which goal reads the contents of two files into the database?',
             'read(filea, fileb)',
             '(tutor, lib].',
             'seeing(sub1, sub2)',
             'No, "read" only has one argument and reads in terms from the input stream.',
             'Either square bracket "[]" notation or the predicate "consult" can be used.',
             'No, "seeing" has one argument which is instantiated to the current input stream.', b),
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Exercise 12 - ARITHMETIC */

a_arith:-newpg,
        write('Exercise 12 - ARITHMETIC'),nl,nl,

        ask('What operator must be present before any arithmetic operators are evaluated?',
            'Which operator will allow an expression like 3+2 to be evaluated?',
            is, 'An "is" assigns the value on its right to the variable on its left.'),

        mult('Which Prolog expression will produce the value "6" ?',
            '3+3',
            'X is 5+2',
            'X is 2+12/3',
            'No, 3+3 is not evaluated. The expression is missing an "is".',
            'No, this is evaluated, but equals "7".',
            '12/3 = 4+2 is 6.  X will be instantiated to "6".', c),
        next2,
        mid_input(_),

        ask('Which operator will produce a remainder?',
            'The remainder of a division is returned by what operator?',
            mod, 'The "mod" operator does an integer division and returns the remainder.'),


        ask('In "Y is 21/5", what is Y instantiated to?',
            'What value is assigned to Y as a result of "Y is 21/5" ?',
            4, 'Integer division with the backslash does not return a remainder.'),

        mult('If "P=7" and "Q=23", which expression will succeed?',
             'P \= Q',
             'P = Q',
             'P >= Q',
             '7 is NOT equal to 27 so this expression will succeed.',
             'No, 7 and 27 are not the same.',
             'No, 7 is not greater than or equal to 27.', a),
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Exercise 14 - BACKTRACKING */

a_backtr:-newpg,
        write('Exercise 14 - BACKTRACKING'),nl,nl,nl,

        mult('What can you do to force backtracking ?',
             'hit a RETURN',
             'type a ";" then a RETURN',
             'nothing',
             'No, just a RETURN will stop the search and never cause backtracking.',
             'The variables in the current goal lose their value and backtracking begins.',
             'No, there is a way to get Prolog to backtrack.', b),
        next2,
        mid_input(_),

        mult('When will Prolog AUTOMATICALLY backtrack ?',
             'when a goal fails',
             'when a match is found',
             'when you hit a RETURN',
             'As soon as a goal fails, Prolog will BACKTRACK automatically.',
             'No, Prolog will try to satisfy the next goal if it finds a match.',
             'No, if Prolog prints a value, RETURN will stop it from resatisfying the goal.', a),
        next2,
        mid_input(_),


        new_db('actor(hoffman).', 'actor(wayne).',
               'actor(bogart).', 'actor(eastwood).'),
        mult('If  "?-actor(G)"  gave you  "G=bogart", what would  "; RETURN"  generate?',
             'G=hoffman',
             'yes',
             'G=eastwood',
             'No, the place-marker is not at the top of the database.',
             'No, assuming G was uninstantiated, it will print a value for G.',
             'The fact "actor(eastwood)." is the next match in the database.', c),
        next2,
        mid_input(_),

        new_db('actress(keaton).', 'actress(minelli).',
               'actress(fonda).', 'actress(monroe).'),
        mult('If X is instantiated to "keaton", the question "?-actress(X)." returns',
             'X=keaton',
             'X=minelli',
             'yes',
             'No, it is trying to match the fact "actress(keaton)." in the database.',
             'No, you are not trying to re-instantiate X.',
             'It successfully matched the fact "actress(keaton)."', c),
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Exercise 9 - CHARACTERS */

a_chars:-newpg,
        write('Exercise 9 - CHARACTERS'),nl,nl,

        ask('In Prolog, is it possible to have a non-printing character? ("y." or "n.")',
            'Is there such a thing as a non-printing character? ("y." or "n.")',
            y, 'A space or end-of-file marker are examples of non-printing characters.'),

        mult('A CHARACTER can also be thought of as',
             'an integer',
             'a variable',
             'a term',
             'Every character, printing and non-printing, has an integer value.',
             'No, a variable name is made up of characters, but a character is constant.',
             'No, each part of a TERM is made up of CHARACTERS.', a),
        next2,
        mid_input(_),

        mult('What is the name of the standard code used to represent characters in Prolog?',
             'SYNTAX',
             'ASCII',
             'TERM',
             'No, syntax is the set of rules for defining terms in Prolog.',
             'It stands for the American Standard Code for Information Interchange.',
             'No, a TERM is a CONSTANT, VARIABLE or STRUCTURE.', b),
        next2,
        mid_input(_),

        mult('Which character is used in the conjunction of goals or to separate arguments?"',
             'the comma',
             'the exclamation mark',
             'the asterisk',
             'A comma separates the arguments of a predicate or ANDs goals in a rule.',
             'No, the exclamation mark is the "CUT" symbol in Prolog.',
             'No, the asterisk is the multiplication operator.', a),
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Exercise 6 - CONJUNCTIONS */

a_conj:-newpg,
        write('Exercise 6 - CONJUNCTIONS'),nl,nl,nl,

        mult('Why would you use a CONJUNCTION ?',
             'to ask a question with multiple goals',
             'to define a fact',
             'to do an arithmetic operation',
             'A conjunction is a question with multiple goals separated by commas.',
             'No, a fact is stored in the database and defines a known relationship.',
             'No, an arithmetic operation is done by the "is" predicate.', a),
        next2,
        mid_input(_),

        mult('What symbol is used to join each of the goals in a conjunction?',
             'a period',
             'a question mark',
             'a comma',
             'No, a period is at the end of the conjunction.',
             'No, a conjunction always starts with a question mark.',
             'A comma between each goal of a conjunction is used to mean "AND".', c),
        next2,
        mid_input(_),

        mult('How would you write "Is Charlie tall and thin ?',
             '?- tall(X), thin(X).',
             '?- tall(charlie), thin(charlie).',
             'tall(charlie):- thin(charlie).',
             'No, this conjunction says "Who is tall and thin?"',
             'We just want to see if "tall(charlie)" and "thin(charlie)" are defined.',
             'No, this is a rule, not a conjunction.', b),
        next2,
        mid_input(_),

        new_db('chops(joe,wood).', 'irons(mary,clothes).',
               'loves(joe,mary).', 'loves(mary,joe).'),
        mult('How would Prolog respond to "?-loves(joe,E), loves(mary,F).',
             'E=wood    F=clothes ',
             'yes',
             'E=mary    F=joe',
             'No, both goals begin with the predicate "loves".',
             'No, the answer "yes" is only returned when a fact is found in the database.',
             'Prolog will find "loves(joe,mary)." and "loves(mary,joe)." in the database.', c),
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Exercise 15 - CUT */

e_cutt:-newpg,
        write('Exercise 15 - CUT'),nl,nl,nl,

        mult('Does CUT represent a ',
             'goal',
             'variable, or',
             'structure',
             'CUT is a goal that always succeeds once and can never be resatified.',
             'No, CUT is represented by an exclamation point and is not a variable.',
             'No, a structure is a term as is a constant or variable. Cut is none of these.', a),
        next2,
        mid_input(_),

        mult('What happens if you backtrack over a CUT?',
             'The previous goal is resatisfied.',
             'It fails.',
             'Prolog starts searching from the top of the database.',
             'No, the previous goal is never reached because you cannot resatify CUT.',
             'An attempt to backtrack over a CUT will always fail.',
             'No, Prolog only starts at the top of the database for every new goal it sees.', b),
        next2,
        mid_input(_),

        mult('Which goals in a rule containing a CUT is a place-marker kept for ?',
             'all of them',
             'none of them',
             'the rules after the CUT',
             'No, there is no need to keep track of the goals that cannot be resatisfied.',
             'No, unless the CUT is the last goal in the rule.',
             'The rules after a CUT can be resatisfied again if necessary.', c),
        next2,
        mid_input(_),

        mult('To denote the CUT goal in a rule, which symbol is used ?',
             'an exclamation point',
             'a backslash',
             'Cut',
             'The exclamation point is the CUT goal.',
             'No, a backslash is used as the integer division operator.',
             'No, "Cut" is a variable. We only say CUT but we write "!".', a),
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Exercise 11 - EQUALITY */

e_equal:-newpg,
        write('Exercise 11 - EQUALITY'),nl,nl,

        ask('Will the goal "dog = cat" succeed? ("y." or "n.")',
            'Will prolog return "yes" for the goal "dog=cat" ? ("y." or "n.")',
            n, 'It will not succeed, "dog" and "cat" are both constants but are not equal.'),

        mult('What type of operator is the "=" ?',
             'prefix',
             'infix',
             'postfix',
             'No, it always appears between two terms.',
             'The equal sign is an infix operator.',
             'No, an equal sign is not used after one argument.', b),
        next2,
        mid_input(_),

        mult('Two structures are NOT equal if',
             'they have the same number of arguments',
             'each corresponding argument is equal',
             'they have different predicates',
             'No, this is one of three requirements two equal structures must meet.',
             'No, this is another requirement two equal structures must meet.',
             'They must have the same predicate and meet the requirements in a) and b).', c),
        next2,
        mid_input(_),

        ask('Will the goal "fish(tuna,large) = fish( G, H )" succeed? ("y." or "n.")',
            'Will prolog return "yes" for that goal ? ("y." or "n.")',
            y, 'It will succeed because all the requirements for two equal structures are met.'),

        mult('What is G in "fish(tuna,large) = fish( G, H )" instantiated to ?',
             'fish',
             'tuna',
             'large',
             'No, fish is the predicate.',
             'G becomes "tuna" because it and "tuna" are the first argument of each goal.',
             'No, H is instantiated to "large".', b),
        next2,
        mid_input(_),

        mult('If the goals "Child=Kid" and "Child=tony" succeed, what happens to "Kid"?',
             'it will also be instantiated to "tony"',
             'it will remain a variable',
             'it will be instantiated to "child"',
             'The first goal causes "Child" and "Kid" to share values.',
             'No, Kid does become instantiated.',
             'No, the constant "child" is not even mentioned here.', a),

        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Exercise 3 - FACTS */

a_facts:-newpg,
        write('Exercise 3 - FACTS'),nl,nl,

        ask('In the sentence "Jack likes Jill." what is the predicate?',
            'Which word describes a relationship between the objects?', likes,
            '"likes" describes the relationship between the objects, "jack" and "jill"'),

        ask('How many objects are in the sentence "Jack likes Jill."? (enter a number)',
            'The predicate "likes" has how many arguments? (enter a number)', 2,
            '"Jack" and "Jill" are the TWO objects or arguments in the sentence.'),

        mult('How would "Jack likes Jill." be written in Prolog?',
             'likes(jack,Jill).',
             'jack(likes,jill)',
             'likes(jack,jill).',
             'No, remember that each part of a FACT must start with a lower case letter.',
             'No, each FACT begins with a predicate, and also ends with a period.',
             'The predicate is "likes" and its arguments are "jack" and "jill".', c),
        next2,
        mid_input(_),

        ask('Are "likes(jack,jill)."   and   "likes(jill,jack)." the same?  ("y." or "n.")',
            'Do they mean the same thing?  ("y." or "n.")', n,
            'The arguments of each fact are in a different order.'),

        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Exercise 13 - LISTS */

e_lists:-newpg,
        write('Exercise 13 - LISTS'),nl,nl,

        mult('What is the correct way to represent a LIST with 3 arguments?',
             '(a,b,c)',
             '[1] [2] [3]',
             '[1, 2, 3]',
             'No, a list is shown in square brackets.',
             'No, this is three separate lists.',
             'A list is in a set of square brackets, each argument separated by a comma.', c),
        next2,
        mid_input(_),

        mult('The "head" of the list  [g, h, i, j, k]  is',
              '[g]',
              'g',
              '[h, i, j, k]',
              'No, this would only be true if the list were [[g], h, i, j, k].',
              'The first element of this list is "g".',
              'No, that is the "tail" of the list.', b),
        next2,
        mid_input(_),

        mult('The list [] is the tail of which list?',
              '[a]',
              '[]',
              '[a, []]',
              'The "head" is "a", which leaves the "tail" to be the empty list.',
              'No, the tail of the empty list will fail.',
              'No, the "tail" here is equal to "[[]]".', a),
        next2,
        mid_input(_),

        new_db('cooks([bacon, eggs, toast]).', 'cooks([lobster],[steamers]).',
               'cooks([dinner]).', 'cooks([breakfast]).'),
        mult('What is Z instantiated to in the question "?-cooks[Y | Z]."',
             'bacon',
             '[toast]',
             '[eggs,toast]',
             'No, Y will be instantiated to "bacon" because it is the head of the list.',
             'No, [toast] is only part of the tail.',
             'Z is the tail, instantiated to a list containing everything but the "head".', c),

        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Exercise 10 - OPERATORS */

e_oper:- newpg,
         writs('Exercise 10 - OPERATORS'),nl,nl,

         ask('What word describes an operator that is written between its arguments?',
             'What type of operator is surrounded by its arguments?',
             infix, 'An INFIX operator, like a "+", is written between two arguments.'),

         ask('Where is a prefix operator in relation to its argument? ("before" or "after")',
             'Does a prefix operator come "before" or "after" its argument?',
             before, 'A PREFIX operator is written before its arguement, like "-3".'),

         mult('Which statement is TRUE ?',
             'addition has a higher precedence than subtraction',
             'division and multiplication have the same precedence',
             'division has a lower precedence than subtraction',
             'No, addition and subtraction have the same precedence.',
             'According to the precedence rules in Prolog, this is true.',
             'No, division has a HIGHER precedence than subtraction.', b),
         next2,
         mid_input(_),

         mult('If you have three operators with the same precedence, how are they evaluated?',
              'right-to-left',
              'at the same time',
              'left-to-right',
              'No, not in Prolog.',
              'No, they are not done at the same time.',
              'Operators with the same precedence are done left to right.', c),
         next2,
         mid_input(_),

         mult('What would be evaluated first in the term  x+y*2/3+2-7  ?',
              'y*2',
              'x+y',
              '2/3',
              'Multiplication is the leftmost operator with the highest precedence.',
              'No, multiplication and division have a higher precedence.',
              'No, it is to the right of an operator with the same precedence.', a),
         (c_level(a) -> next2; next3),
         end_input.
```

```
/* Exercise 4 - QUESTIONS */

a_quest:-newpg,
        write('Exercise 4 - QUESTIONS'),nl,nl,

        mult('Which one of these is a valid QUESTION ?',
             '?- eats(harry,lunch).',
             '?- (mary,john,cousins)',
             '? _knits(irene,sweaters).',
             'This is a valid question. It is a FACT that begins with the "?-" symbols. ',
             'No, there are 3 arguments but no predicate.',
             'No, it does not begin with "?-" and the predicate "_knits" is a variable.', a),
        next2,
        mid_input(_),

        new_db('brown(pants).','white(shirt).','yellow(tie).','tan(jacket).'),
        mult('Which question would PROLOG answer "yes" to ?',
             '?- red(socks).',
             '?- brown(pants).',
             '?- white(jacket).',
             'No, the fact "red(socks)." is not in the database.',
             'The fact "brown(pants)." is in the database.',
             'No, the fact "white(jacket)." is not in the database.', b),
        next2,
        mid_input(_),

        mult('How would you ask PROLOG "Did John paint the house ?"',
             '?- (john),paint(house).',
             '?- John(paint,house).',
             '?- paint(john,house).',
             'No, it has an incorrect QUESTION format.',
             'No, the predicate "John" is a variable.',
             'The predicate is "paint" and its objects are "john" and "house".', c),
        next2,
        mid_input(_),

        mult('Which of these would NOT be found in a Prolog database?',
             'sleeps(baby,soundly).',
             '?- queen(elizabeth,england).',
             'brother.',
             'No, facts are stored in the database.',
             'This is a question and they are used to query the database.',
             'No, this is a fact made up of a predicate and no arguments.', b),
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Exercise 7 - RULES */

e_rules:-newpg,
        write('Exercise 7 - RULES'),nl,nl,

        ask('Are RULES stored in the database?  ("y." or "n.")',
             'Can you consult a file that contains rules ?  ("y." or "n.")', y,
             'Rules, like facts, are stored in the database.'),

        mult('How many goals must be satisfied before the whole rule is proven true ?',
              'none of them',
              'all of them',
              'half of them',
              'No, each time a goal fails something in the rule is not true. ',
              'Every goal in a rule must be true to make the whole rule true.',
              'No, only some true goals do not prove anything.', b),
        next2,
        mid_input(_),

        mult('What do the symbols ":-" , between the head and body, stand for ?',
              'IF',
              'equals',
              'AND',
              'The head of a rule is true, IF all the goals in the body are true.',
              'No, an equal sign is written "=" .',
              'No, a comma is used to represent AND in Prolog.', a),
        next2,
        mid_input(_),

        mult('When is the head of a rule proven true ?"',
              'never',
              'when its variable arguments are all instantiated',
              'after the subgoals in the body of the rule are satisfied',
              'No, the head of a rule is true when its subgoals are proven true.',
              'No, its truth depends on the subgoals, not its arguments.',
              'IF every goal in the body is matched, the parent goal is true.', c),
        (c_level(s) -> next2; next3),
        end_input.
```

```
/* Exercise 17 - READING/WRITING CHARACTERS */

e_rwchar:-newpg,
        write('Exercise 17 - READING/WRITING CHARACTERS'),nl,nl,nl,

        mult('Which predicate will read a space (ASCII 32)  ?',
             'get',
             'read',
             'get0',
             'No, "get" skips non-printing characters.',
             'No, "read" is always looking for a period before a space.',
             'The predicate "get0" will read both printing and non-printing characters.', c),
        next2,
        mid_input(_),

        mult('If A is instantiated to "b", in which case will "get(A)" succeed?',
             'the next character is a space',
             'the next character is a "b"',
             'the next character is an "a"',
             'No, "get" will not see the space.',
             'No, since A is set to "b", it is looking for it, this will pass.',
             'It will fail here because "get" wants to find a "b" in the next character.', c),
        next2,
        mid_input(_),

        mult('What will NOT happen if you backtrack over a "put" ?',
             'it will pass',
             'it will reprint its character again',
             'it will fail',
             'It will NOT pass during backtracking.',
             'No, a side effect of trying to resatisfy a "put" is just this.',
             'No, a failure will definitely occur.', a),
        next2,
        mid_input(_),

        mult('What is the predicate "tab" used for ?',
             'to print spaces',
             'to output tabs',
             'to cause backtracking',
             'The goal "tab(20)" will print out 20 spaces.',
             'No, Prolog does not have an actual "tabbing" function.',
             'No, only the "fail" goal, input ";  (RETURN)" or a "no match" condition will.', a),
        next2,
        mid_input(_),

        ask('Will "tab(two)" pass? ("y." or "n.")',
            'Would the goal "tab(two)" succeed ("y." or "n.")',
             n, 'The argument of "tab" must be an integer.'),


        mult('Where will "skip(p)" stop in the input stream " a.cpom12 " ?',
             'at the end-of-input character',
             'at the "o"',
             'after the period',
             'No, "skip(p)" is looking for the character "p".',
             'After "skip" finds the "p", it stops at the next character.',
             'No, not unless the goal were "skip(.)."', b),


        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Exercise 16 - READING/WRITING TERMS */

e_rwterm:-newpg,
        write('Exercise 16 - READING/WRITING TERMS'),nl,nl,nl,

        mult('The initial and most common input/output stream is ',
             'the Prolog database',
             'a file',
             'the terminal',
             'No, the database is used to store facts and rules. I/O is not done there.',
             'No, files can be used for input and output, but another means is more common.',
             'Most of your input and output will probably be done with a terminal.', c),
        next2,
        mid_input(_),

        mult('The goal "read(Next)" and terminal input "good. morning." would set Next to',
             'good',
             'good  morning',
             'morning',
             'It would take another "read" to pick up "morning".',
             'No, "read" stops when it sees the period and space.',
             'No, Prolog sees "good" first.', e),
        next2,
        mid_input(_),

        ask('If you backtrack over a "read", is the next term read ? ("y." or "n.")',
            'Can "read" be re-satisfied ? ("y." or "n.")',
            n, 'The "read" goal will not reinstantiate its argument more than once.'),

        mult('After the goal "X=20", what would "write(X)" output ?',
             '_321',
             '20',
             'X=20',
             'No, X is instantiated so an internal representation would not be shown.',
             'A "20" is output because X was set to that value in the previous goal.',
             'No, just the value of X is written.', b),
        next2,
        mid_input(_),

        mult('Which goal would output  GOOD MORNING  onto the screen ?',
             'write(  GOOD MORNING  )',
             'out("  GOOD MORNING  ")',
             'neither goal would work',
             'No, write needs its arguments in quotes.',
             'No, Prolog does not have an "out" predicate.',
             '"write" expects it argument in single quotes.'
, c),
        (c_level(e) -> next2; next3),
        and_input.
```

```
/* Exercise 8 - SYNTAX */

e_syntax:-newpg,
        write('Exercise 8 - SYNTAX'),nl,nl,

        ask('Is a FACT with correct syntax considered to be a STRUCTURE? ("y." or "n.")',
            'Is a FACT a STRUCTURE as opposed to being an ATOM or INTEGER? ("y." or "n.")',
            y, 'STRUCTURES consist of a predicate and arguments as do FACTS.'),

        mult('Which of these is NOT considered to be a CONSTANT ?',
            'an atom',
            'an integer',
            'a structure',
            'No, an atom is a CONSTANT.',
            'No, integers are CONSTANTS.',
            'A structure can contain variables, thus it is not CONSTANT.', c),
        next2,
        mid_input(_),

        mult('Are the symbols "?-" and ":-" considered to be',
             'variables',
             'structures, or',
             'atoms',
             'No, they do not begin with a capital letter or an underscore.',
             'No, they are not structures because they do not have a predicate or arguments',
             'These symbols and others like the comma, colon and semi-colon are atoms.', c),
        next2,
        mid_input(_),

        mult('Which of the following is NOT an ATOM ?"',
            '!&()',
            '123abc',
            'def05482',
            'No, a string of all symbols is considered to be an ATOM.',
            'A string beginning with a number, must be in single quotes to be an ATOM.',
            'No, it starts with a small letter and contains only letters and numbers.', b),
        next2,
        mid_input(_),

        mult('Every TERM in Prolog is a sequence of',
            'variables',
            'characters',
            'integers',
            'No, a variable is a TERM.',
            'Everything in Prolog is a TERM that consists of CHARACTERS.',
            'No, an integer is a CONSTANT which is really a TERM.', b),
        next2,
        mid_input(_),

        mult('What does the symbol "_" , by itself, mean in Prolog?',
            'IF',
            'the anonymous variable',
            'OR',
            'No, IF is written as ":-" within a rule.',
            'The anonymous variable can be used as one of the arguments of a predicate.',
            'No, the symbol ";" means OR in Prolog.', b),
        (c_level(s) -> next2; next3),
        end_input.
```

```
/* Exercise 5 - VARIABLES */

s_vars:-newpg,
        write('Exercise 5 - VARIABLES'),nl,nl,nl,

        mult('How can you tell if something is a VARIABLE in Prolog ?',
             'It begins with a question mark.',
             'It starts with a capital letter.',
             'It ends with a period.',
             'No, a QUESTION always begins with a question mark and hyphen.',
             'A VARIABLE always begins with a capital letter or an underscore.',
             'No, every FACT or RULE in Prolog must end with a period.', b),
        next2,
        mid_input(_),

        mult('What does it mean when a VARIABLE is INSTANTIATED?',
             'It loses its value.',
             'It has no value.',
             'It is set equal to a new value.',
             'No, a VARIABLE loses its value when you try to resatisfy the goal it is in.',
             'No, initially every variable has no value or is UNINSTANTIATED.',
             'A VARIABLE is INSTANTIATED each time it takes on a new value.', c),
        next2,
        mid_input(_),

        new_db('sister_of(julie,dennis).','brother_of(dennis,tom).',
               'sister_of(julie,tom).','sister_of(mary,tom).'),
        mult('How would Prolog respond to "?- sister_of(julie,Y)."',
             'yes',
             'Y=dennis',
             'Y=tom',
             'No, since the variable is named Y, Prolog will tell you what Y equals.',
             'The first "sister_of" rule in the database is a match for this question.',
             'No, Prolog would find the first "sister_of" rule before this one.', b),
        next2,
        mid_input(_),

        new_db('sister_of(julie,dennis).','brother_of(dennis,tom).',
               'sister_of(julie,tom).','sister_of(mary,tom).'),
        mult('For "?- sister_of(julie,Y).", you get "Y=dennis". Typing a ";  {RETURN}" gives',
             'Y=tom',
             'Y=dennis',
             'no',
             '"sister_of(julie,tom) is the next match found. Y is instantiated to "tom".',
             'No, the place marker starts the search after this fact.',
             'No, Prolog can find a second match.', a),
        (c_level(s) -> next2; next3),
        end_input.
```

```
/* Lesson 11 - EQUALITY */

equal:- newpg,
        write('Lesson 11 - EQUALITY'),nl,nl,nl,
        write('EQUALITY is a built-in predicate of PROLOG that is represented by the'),nl,nl,
        write('infix operator "=".  Every variable is assumed to equal itself.  When'),nl,nl,
        write('we write             car(X):- has_four_wheel(X),'),nl,
        write('                              has_engine(X),'),nl,
        write('                              has_body(X).'),nl,nl,
        write('there is no need to write "X=X.", it is already assumed by PROLOG.  Each X in'),nl,nl,
        write('in this rule is the same and if any occurence of it becomes instantiated, each'),nl,nl,
        write('X will be set to that value.  Integers and atoms are always equal to themselves.'),nl,nl,
        write('         "flowers=flowers"      will succeed'),nl,
        write('         "hamburger = hotdog"   will fail'),nl,
        write('         "123456 = 123456"      will succeed'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('EQUALITY - continued'),nl,nl,nl,
        write('If A is any object and B is an uninstantiated variable, "A=B." will succeed'),nl,nl,
        write('and B will be instantiated to whatever A is.  For example, the goal '),nl,nl,
        write('"likes(john,skydiving) = B." will succeed and B gets instantiated to'),nl,nl,
        write('the string  "likes(john,skydiving)".'),nl,nl,
        write('Two structures are EQUAL if all of the following are true :'),nl,nl,
        write('         - they both have the same predicate'),nl,
        write('         - they both have the same number of arguments, and'),nl,
        write('         - each corresponding argument is equal.'),nl,nl,
        write('The goal "rakes(dave,leaves) = rakes(X,Y)" will succeed.  X and Y are'),nl,nl,
        write('instantiated to "dave" and "leaves", respectively.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('EQUALITY - continued'),nl,nl,nl,
        write('With two uninstantiated variables, a goal like "Alpha = Beta." will succeed'),nl,nl,
        write('and cause the variables to SHARE.  When either one is instantiated to a'),nl,nl,
        write('value, the other one will also get instantiated.'),nl,nl,nl,
        write('Another predicate, NOT EQUAL, is represented with the symbols "\=".'),nl,nl,
        write('If the goal "ABC \= DEF." succeeds, then "ABC = DEF."  will fail and if'),nl,nl,
        write('the goal "ABC = DEF." succeeds, then "ABC \= DEF." will fail.'),nl,nl,nl,nl,
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Experienced level Menu ecreen end dielog */

exper:-newpg,
       write(' You ere in the experienced level of TUTOR.  If the menu rapeats immediataly, '),nl,nl,
       write(' an INVALID INPUT wae received.  The PROLOG topics ara:'),nl,nl,nl,nl,
       menu,nl,
       write('Choosa a topic by typing ite number followed '),nl,nl,
       writa('by a period and a RETURN  (ie. "6."  (RETURN) )'),nl,nl,
       next1,
       mid_input(Num1),
       (integer(Num1) -> get_it(Num1); exper),
       exper.
```

```
/* Lesson 3 - FACTS */

facts:- newpg,
        write('Lesson 3 - FACTS'),nl,nl,nl,
        write('The sentence "Kids like candy", would be written as "like(kids,candy)."'),nl,nl,
        write('in PROLOG.  A declarative statement like this is called a FACT.  A FACT is'),nl,nl,
        write('made up of one predicate and zero or more objects.  This particular FACT '),nl,nl,
        write('consists of two objects, "kids" and "candy" and a predicate "like", '),nl,nl,
        write('which is the relationship between the objects.'),nl,nl,
        write('Each element of a FACT is a constant and must begin with a lower-case '),nl,nl,
        write('letter. The predicate, also called a functor, is always written first.'),nl,nl,
        write('The objects, also called arguments, are separated by commas and enclosed '),nl,nl,
        write('in a pair of parentheses.  A FACT is always terminated by a period.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('FACTS - continued'),nl,nl,nl,
        write('Here are some examples of FACTS and their English equivalents:'),nl,nl,nl,
        write('   dangerous(tornedos).         "Tornados are dangerous" '),nl,nl,
        write('   plays(john,soccer,well).     "John plays soccer well" '),nl,nl,
        write('   male(charlie).               "Charlie is a male" '),nl,nl,
        write('   greek(susan).                "Susan is Greek" '),nl,nl,
        write('   joined(harold,army).         "Harold joined the Army" '),nl,nl,
        write('   father_of(maurice,leslie).   "Maurice is the father of Leslie" '),nl,nl,nl,
        write('Each FACT entered into PROLOG becomes part of the current database.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('FACTS - continued'),nl,nl,nl,
        write('When creating FACTS, the ordering of the arguments can be very important.'),nl,nl,
        write('These FACTS were written to describe different vegetables.'),nl,nl,
        write('         vegetable(lettuce,green).'),nl,
        write('         vegetable(carrot,orange).'),nl,
        write('         vegetable(beet,red).'),nl,nl,
        write('Notice that the first argument is used for the vegetable name and the'),nl,nl,
        write('second for its color.  The FACT "vegetable(red,radish)." does not follow'),nl,nl,
        write('the same conventions.  If the database was searched for all'),nl,nl,
        write('red vegetables, each FACT with "red" as its second argument, this fact'),nl,nl,
        write('would not be found.'),nl,nl,nl,
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Clauses that choose a particular Prolog topic */

get_it(1):- asserta(c_topic(1)),
            topic('/usr/prolog/hist', hist, 0),
            retract(c_topic(1)).
get_it(2):- asserta(c_topic(2)),
            topic('/usr/prolog/cpro', cpro, 0),
            retract(c_topic(2)).
get_it(3):- asserta(c_topic(3)),
            topic('/usr/prolog/facts', facts, 0),
            topic('/usr/prolog/s_facts', s_facts, 0),
            topic('/usr/prolog/e_facts', e_facts, 0),
            retract(c_topic(3)).
get_it(4):- asserta(c_topic(4)),
            topic('/usr/prolog/quest', quest, 0),
            topic('/usr/prolog/s_quest', s_quest, 0),
            topic('/usr/prolog/e_quest', e_quest, 0),
            retract(c_topic(4)).
get_it(5):- asserta(c_topic(5)),
            topic('/usr/prolog/vars', vars, 0),
            topic('/usr/prolog/s_vars', s_vars, 0),
            topic('/usr/prolog/e_vars', e_vars, 0),
            retract(c_topic(5)).
get_it(6):- asserta(c_topic(6)),
            topic('/usr/prolog/conj', conj, 0),
            topic('/usr/prolog/s_conj', s_conj, 0),
            topic('/usr/prolog/e_conj', e_conj, 0),
            retract(c_topic(6)).
get_it(7):- asserta(c_topic(7)),
            topic('/usr/prolog/rules', rules, 0),
            topic('/usr/prolog/s_rules', s_rules, 0),
            topic('/usr/prolog/e_rules', e_rules, 0),
            retract(c_topic(7)).
get_it(8):- asserta(c_topic(8)),
            topic('/usr/prolog/syntax', syntax, 0),
            topic('/usr/prolog/s_syntax', s_syntax, 0),
            topic('/usr/prolog/e_syntax', e_syntax, 0),
            retract(c_topic(8)).
get_it(9):- asserta(c_topic(9)),
            topic('/usr/prolog/chars', chars, 0),
            topic('/usr/prolog/s_chars', s_chars, 0),
            topic('/usr/prolog/e_chars', e_chars, 0),
            retract(c_topic(9)).
```

```
get_it(10):- asserta(c_topic(10)),
             topic('/usr/prolog/oper', oper, 0),
             topic('/usr/prolog/s_oper', s_oper, 0),
             topic('/usr/prolog/e_oper', e_oper, 0),
             retract(c_topic(10)).
get_it(11):- asserta(c_topic(11)),
             topic('/usr/prolog/equal', equal, 0),
             topic('/usr/prolog/s_equal', s_equal, 0),
             topic('/usr/prolog/e_equal', e_equal, 0),
             retract(c_topic(11)).
get_it(12):- asserta(c_topic(12)),
             topic('/usr/prolog/arith', arith, 0),
             topic('/usr/prolog/s_arith', s_arith, 0),
             topic('/usr/prolog/e_arith', e_arith, 0),
             retract(c_topic(12)).
get_it(13):- asserta(c_topic(13)),
             topic('/usr/prolog/lists', lists, 0),
             topic('/usr/prolog/s_lists', s_lists, 0),
             topic('/usr/prolog/e_lists', e_lists, 0),
             retract(c_topic(13)).
get_it(14):- asserta(c_topic(14)),
             topic('/usr/prolog/backtr', backtr, 0),
             topic('/usr/prolog/s_backtr', s_backtr, 0),
             topic('/usr/prolog/e_backtr', e_backtr, 0),
             retract(c_topic(14)).
get_it(15):- asserta(c_topic(15)),
             topic('/usr/prolog/cutt', cutt, 0),
             topic('/usr/prolog/s_cutt', s_cutt, 0),
             topic('/usr/prolog/e_cutt', e_cutt, 0),
             retract(c_topic(15)).
get_it(16):- asserta(c_topic(16)),
             topic('/usr/prolog/rwterm', rwterm, 0),
             topic('/usr/prolog/s_rwterm', s_rwterm, 0),
             topic('/usr/prolog/e_rwterm', e_rwterm, 0),
             retract(c_topic(16)).
get_it(17):- asserta(c_topic(17)),
             topic('/usr/prolog/rwchar', rwchar, 0),
             topic('/usr/prolog/s_rwchar', s_rwchar, 0),
             topic('/usr/prolog/e_rwchar', e_rwchar, 0),
             retract(c_topic(17)).
get_it(18):- asserta(c_topic(18)),
             topic('/usr/prolog/accfile', accfile, 0),
             topic('/usr/prolog/s_accfile', s_accfile, 0),
             topic('/usr/prolog/e_accfile', e_accfile, 0),
             retract(c_topic(18)).
get_it(19):- asserta(c_topic(19)),
             topic('/usr/prolog/builtin', builtin, 0),
             retract(c_topic(19)).
get_it(_).
```

```
/* Lesson 1 - some History on PROLOG */

hist:-  newpg,
        write('Lesson 1 - The History of PROLOG'),nl,nl,nl,
        write('PROLOG was originally developed in 1972 by Colmerauer and Roussel at the '),nl,nl,
        write('University of Marseilles as a practical tool for PROgramming in LOGic.'),nl,nl,
        write('It was the first interpreter of its kind and was created about the same time'),nl,nl,
        write('people were discovering that logic sentences could be expresssed as program'),nl,nl,
        write('statements and that controlled inference was analogous to the execution of'),nl,nl,
        write('these statements.'),nl,nl,
        write('Since its creation, there has been a considerable proliferation of PROLOG'),nl,nl,
        write('implementations that cover a wide range of design philosophies, host machines,'),nl,nl,
        write('and application environments. '),nl,nl,nl,
        next2,
        mid_input(_),
        not_var(R1,R2),
        quit(R2),
        newpg,
        write('Brief Overview of PROLOG'),nl,nl,nl,
        write('A program written in PROLOG describes known facts and relationships about'),nl,nl,
        write('a problem, whereas, in other languages, a program prescribes a sequence'),nl,nl,
        write('of steps that must be taken by the computer to solve a problem.'),nl,nl,
        write('Programming in PROLOG can be thought of as three basic steps:'),nl,nl,
        write('          Declaring FACTS '),nl,
        write('          Defining RULES  and'),nl,
        write('          Asking QUESTIONS '),nl,nl,
        write('Prolog is an interpreter with a database that you populate, depending'),nl,nl,
        write('on your application.  Implementations of PROLOG on conventional computers'),nl,nl,
        write('have reached efficiency comparable to pure LISP.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('Overview of PROLOG - continued'),nl,nl,nl,
        write('Prolog is used in many different fields.  Areas like'),nl,nl,
        write('          ARTIFICIAL INTELLIGENCE'),nl,
        write('          COMPILER CONSTRUCTION'),nl,
        write('          RELATIONAL DATABASES'),nl,
        write('          MATHEMATICAL LOGIC'),nl,
        write('          NATURAL LANGUAGE PROCESSING'),nl,
        write('          ABSTRACT PROBLEM SOLVING  and'),nl,
        write('          EXPERT SYSTEMS'),nl,nl,
        write('are common applications.   The Japanese have chosen to use Prolog as'),nl,nl,
        write('the kernal language in the Fifth-generation computers.  One reason is'),nl,nl,
        write('because it does not presuppose a Von Neumann architecture like most other'),nl,nl,
        write('programming languages.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('Overview of PROLOG - continued'),nl,nl,nl,
        write('One of the major attractions to PROLOG is its ease of programming.  PROLOG'),nl,nl,
        write('programs consist of clauses, in first-order logic and a theorem to be proven.'),nl,nl,
        write('Logic programming is based on clauses of the form,  "H if P and Q".'),nl,nl,
        write('These are called "Horn clauses".  Problem "H" can be reduced to subproblems'),nl,nl,
        write('"P" and "Q".  Prolog has many advantages as an application language primarily'),nl,nl,
        write('because of its powerful pattern-matching capabilities that come from'),nl,nl,
        write('unification and automatic backtracking.  A main criticism of PROLOG though,'),nl,nl,
        write('is that its left-to-right, depth-first search strategy is too rigid. It can'),nl,nl,
        write('be inefficient and incomplete for certain applications.'),nl,nl,
        next2,
        end_input.
```

```
/* Intermediate level - choice of lesson or menu */


interm:- newpg,
        write('This level assumes you are familiar with PROLOG or have used this tool'),nl,nl,
        write('before.  The lessons are presented in the following order:'),nl,nl,nl,
        menu,
        write('Do you want to continue from a certain point  OR  do you want to select'),nl,nl,
        write('a particular topic? ("c."(ontinue) / "s."(elect))'),nl,nl,
        next1,
        mid_input(Ans1),
        c_or_s(Ans1).

c_or_s(s):- write('Enter the number of the lesson you want to review,'),nl,nl,
           write('followed by a period and a RETURN  (ie. "6."  (RETURN) )'),nl,
           next1,
           mid_input(Lnum4),
           (integer(Lnum4) -> asserta(c_level(a)),
                              Lnum5 is Lnum4 + 1,
                              seq(Lnum4,Lnum5) ; newpg,
                                                 bad_inp,
                                                 menu,
                                                 c_or_s(s)),
           topic(expar,0).

c_or_s(_):- write('Type in the lesson number you want to start at, followed by a'),nl,nl,
           write('period and a RETURN  (ie. "6."  (RETURN) )'),nl,nl,
           next1,
           mid_input(Lnum2),
           (integer(Lnum2) -> asserta(c_level(b)),
                              max(M),              /* from "start" */
                              seq(Lnum2,M); newpg,
                                            bad_inp,
                                            menu,
                                            c_or_s(c)).
```

```
/* Librery of common routinee */

                                /* If Reply1 ie e verieble, "bad_reply" is
                                   returned.  Else, Reply1 ie untouched */

not_ver(Reply1,bad_reply):- ver(Reply1).
not_ver(Reply1,Reply1).

topic( X, Y, Z ):- consult(X), Y, ebolish(Y,Z).

repeet_topic(r):- repeett.              /* To repeet e topic */
repeet_topic(_).

seq(X,Y):- X \== Y,
           get_it(X),
           X1 Te X + 1,
           eeq(X1,Y).

seq(_,_):- c_level(e).

seq(_,_):- new_lev(n).

                                /* Hendlee input between screens of e lesson,
                                   eummary or exercise. Aleo input to ekill
                                   level questions  */

mid_input(R2) :- read(R1),
                 not_ver(R1,R2),
                 quit(R2).

mid_input(R2) :- R2 = 'BAD SYNTAX - Anewer'.

                                /* Hendles input efter the lest screen of
                                   e leseon, eummary or exerciee. */

end_input:- read(R3),
            not_ver(R3,R4),
            repeet_topic(R4),
            quit(R4).
end_input.


quit(helt):- level_stop.        /* Rulee to leeve TUTOR */
quit(q):- level_etop.
quit(quit):- level_etop.
quit(exit):- level_etop.
quit(_).
```

```
level_stop:- nl,nl,
             write('Do you went to try e new skill level? ("y." / "n.")'),
             nl,nl,nl,
             mid_input(Ane1),
             new_lev(Ans1).

new_lev(y):- ebolish(show_db,0),
             write('Which new level do you went?'),nl,nl,
             write('    Beginner     -- type "b.  (RETURN)" '),nl,
             write('    Intermediete -- type "i.  (RETURN)"'),nl,
             write('    Experienced  -- type "e.  (RETURN)"'),nl,nl,nl,
             next1,
             mid_input(L1),
             c_level(C),
             retrect(c_level(C)),
             level(L1).                  /* level defined in stert file */


new_lev(_):- newpg,
             write('This session of the PROLOG tutoriel hes ended.'),nl,nl,
             write('Feel free to use it egein enytime for e refresher'),nl,nl,
             write('or to continue from where you left off.'),nl,nl,nl,nl,nl,
             helt.

new_db(I, J, K, L):- ebolish(db1,1),
                     ebolish(db2,1),
                     ebolish(db3,1),
                     ebolish(db4,1),
                     esserte(show_db),
                     esserte(db1(I)),
                     esserte(db2(J)),
                     esserte(db3(K)),
                     esserte(db4(L)).

print_db:- write('Given the following detebase: '),
          db1(D1), write(D1),nl,
          db2(D2), teb(30), write(D2),nl,
          db3(D3), teb(30), write(D3),nl,
          db4(D4), teb(30), write(D4),nl,nl.


bad_inp:- nl,write('INVALID INPUT - TRY AGAIN'),nl,nl,nl,nl.

newpg:- nl, nl, nl, nl, nl, nl, nl, nl, nl, nl, nl, nl,
        nl, nl, nl, nl, nl, nl, nl, nl, nl, nl, nl, nl.

/* Menu screen */

menu:-  write(' 1) history of PROLOG    7) rules                  13) lists'),nl,
        write(' 2) C-PROLOG et KSU      8) syntex                 14) backtrecking '),nl,
        write(' 3) fects                9) cherecters             15) cut'),nl,
        write(' 4) questions            10) operetors   '),nl,
        write(' 5) veriebles            11) equelity'),nl,
        write(' 6) conjunctions 12) erithmetic'),nl,nl,
        write(' 16) reeding/writing cherecters          18) eccessing files'),nl,
        write(' 17) reading/writing terms               19) built-in predicetes'),nl,nl,nl.


next1:- write('                                              TO QUIT - type "q."'),nl.
next2:- write('                          NEXT SCREEN  - type "n."   TO QUIT - type "q."'),nl.
next3:- write('TO REPEAT A TOPIC - type "r."  NEXT SCREEN  - type "n."   TO QUIT - type "q."'),nl.
```

```
/* Lesson 13 - LISTS */

lists:- newpg,
        write('Lesson 13 - LISTS'),nl,nl,nl,
        write('A LIST is an ordered sequence of elements that can be of any length.  The'),nl,nl,
        write('elements can be atoms, structures or any other term including other lists.'),nl,nl,
        write('A LIST is either EMPTY, containing no elements, or it is a structu '),nl,nl,
        write('with two parts: a "head" and a "tail".  The EMPTY list is written "[]".'),nl,nl,
        write('The easiest notation for writing LISTS is shown in these examples.'),nl,nl,nl,
        write('           []      [1,2,3]          [a,b,c,(d,e],F,G]'),nl,nl,nl,
        write('Every element is separated by a comma and the entire list is enclosed in'),nl,nl,
        write('square brackets.  Each list inside a list follows the same conventions.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('LISTS - continued'),nl,nl,nl,
        write('Each of the elements in a LIST are accessible when the list is split up'),nl,nl,
        write('into a "head" and "tail".  The head is the first argument of the list and'),nl,nl,
        write('the tail is the rest of the list.  This is very similar to the way LISP'),nl,nl,
        write('handles lists.  Notice the examples below.'),nl,nl,nl,
        write('         LIST            HEAD            TAIL'),nl,
        write('         ----            ----            ----'),nl,
        write('       [1,2,3,4]          1             [2,3,4]'),nl,
        write('         []             (fails)         (fails)'),nl,
        write('       [[a,b],c]         [a,b]            [c]'),nl,
        write('        [x,[a]]           x              [[a]]'),nl,
        write('         [[a]]           [a]              [] '),nl,nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('LISTS - continued'),nl,nl,nl,
        write('To represent a LIST with a head X and a tail Y, you would write "[X|Y]".'),nl,nl,
        write('The separator is a verticle bar.  This notation instantiates X to the'),nl,nl,
        write('head of the LIST and Y to the tail as shown in the example.'),nl,nl,nl,
        write('DATABASE:        eats([popcorn,peanuts,candy]).'),nl,
        write('                 eats([dinner,[meat,potatoes]]).'),nl,nl,nl,
        write('in PROLOG:       ?- eats([A|B]).  (RETURN)'),nl,
        write('                   ----------------------'),nl,
        write('                 A = popcorn  B = [peanuts,candy] ;  (RETURN)'),nl,
        write('                                                   ----------'),nl,
        write('                 A = dinner   B = [[meat,potatoes]]'),nl,nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('LISTS - continued'),nl,nl,nl,
        write('To search an entire LIST for a particular atom, each element in the list'),nl,nl,
        write('would have to be examined.  This can be done using a recursive algorithm.'),nl,nl,
        write('                1) Look at the head of the list'),nl,nl,
        write('                2) Is this the atom?  If YES - STOP'),nl,
        write('                                      If NO  - continue'),nl,nl,
        write('                3) The tail becomes the new list'),nl,nl,
        write('                4) REPEAT'),nl,nl,
        write('PROLOG has a built-in function called "member" that will execute this'),nl,nl,
        write('sequence for you.  "Member" is covered in the lesson on built-in predicates.'),nl,nl,nl,nl,
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Multiple-choice routine that asks a question, then gives
   choices to pick the correct answer from.  */


mult(Q, A, B, C, RA, RB, RC, Ans):- asserta(quest(Q)),
                                    asserta(ra(RA)),
                                    asserta(rb(RB)),
                                    asserta(rc(RC)),
                                    asserta(ans(Ans)),
                                    m_out(A, B, C, 1),
                                    retract(ans(Ans)),
                                    retract(rc(RC)),
                                    retract(rb(RB)),
                                    retract(ra(RA)),
                                    retract(quest(Q)),
                                    abolish(show_db,0).


m_out(A, B, C, X):- (show_db -> nl, print_db, quest(Q); quest(Q)),
                    nl,nl,write(Q),nl,nl,
                    tab(5),write('a)  '),write(A),nl,nl,
                    tab(5),write('b)  '),write(B),nl,nl,
                    tab(5),write('c)  '),write(C),nl,nl,
                    write('            Choose an answer by its letter (ie. "a.")'),nl,nl,
                    next1,
                    mid_input(R1),nl,
                    answer(R1, X, A, B, C).

answer(R,_,_,_,_):- ans(R),
                    write('Yes, '), reas(R).

answer(R,0,_,_,_):- reas(R),
                    ans(Y), write('The answer is '), write(Y), put(46),nl,
                    reas(Y).

answer(R,0,_,_,_):- ans(Y), write('The answer is '), write(Y), put(46),nl,
                    reas(Y).

answer(R,Z,A,B,C):- Z1 is Z - 1,
                    reas(R),
                    m_out(A, B, C, Z1).

answer(R,Z2,A,B,C):- write('Please choose "a.", "b." or "c."'),nl,nl,
                     Z3 is Z2 - 1,
                     m_out(A, B, C, Z3).

reas(a):- ra(RA),
          write(RA),nl,nl,nl.

reas(b):- rb(RB),
          write(RB),nl,nl,nl.

reas(c):- rc(RC),
          write(RC),nl,nl,nl.
```

```
/* Lesson 10 - OPERATORS */

oper:- newpg,
       write('Lesson 10 - OPERATORS'),nl,nl,nl,
       write('Every OPERATOR in PROLOG has three properties: its position, precedence'),nl,nl,
       write('and its associativity.'),nl,nl,nl,
       write('The OPERATORS  plus "+", minus "-", multiply "*" and divide "/" are written'),nl,nl,
       write('between their arguments and thus called "infix" OPERATORS.  When a minus "-"'),nl,nl,
       write('sign is put before its argument to denote a negative number, it is called'),nl,nl,
       write('a "prefix" OPERATOR.  An OPERATOR written after its argument is called'),nl,nl,
       write('"postfix".  The position of an OPERATOR, therefore, is where it is written'),nl,nl,
       write('in relation to its arguments.'),nl,nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('OPERATORS - continued'),nl,nl,nl,
       write('When PROLOG evaluates an expression like "x+y*2/3-z", it must use OPERATOR'),nl,nl,
       write('precedence and associativity.  Each OPERATOR has a precedence class assigned'),nl,nl,
       write('to it, and the one with the highest precedence is always evaluated first.'),nl,nl,
       write('Multiplication and division have a higher precedence than addition and'),nl,nl,
       write('subtraction.  All four operations are left-associative which means OPERATORS'),nl,nl,
       write('with the same precedence are evaluated left-to-right.  Parentheses are often'),nl,nl,
       write('used for clarity or to override an automatic precedence.'),nl,nl,nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('OPERATORS - continued'),nl,nl,nl,
       write('Rewriting the expression  "x+y*2/3-z" with parentheses would look like:'),nl,nl,
       write('          ((x +((y * 2)/3))- z)'),nl,nl,
       write('and would read: "Multiply y by 2 first, then divide that result by 3, then'),nl,nl,
       write('add that result to x.  Then subtract z from that result."'),nl,nl,nl,
       write('Any term containing an arithmetic operator is NOT evaluated unless it is'),nl,nl,
       write('the argument to the predicate "is".  This is covered in the ARITHMETIC lesson.'),nl,nl,nl,nl,nl,
       (c_level(a) -> next2; next3),
       end_input.
```

```
/* Lesson 4 - QUESTIONS */

queet:- newpg,
        write('Lesson 4 - QUESTIONS'),nl,nl,nl,
        write('QUESTIONS in Prolog look very eimilar to FACTS, but they ere not etored'),nl,nl,
        write('in the datebaee.  Every QUESTION begins with e "?-" .   For example,'),nl,nl,nl,
        write(' "John eete peee."        ie written      "eets(john,peee)."'),nl,nl,
        write(' "Does John eat pees?"   ie written    "?-eats(john,pees)."'),nl,nl,nl,
        write('Whenever PROLOG sees e QUESTION, it eearchea the current datebase, from '),nl,nl,
        write('the top, for e FACT to match.  A match only occure when tha predicate end'),nl,nl,
        write('every ergument of the QUESTION is identicel to e FACT in the database.'),nl,nl,
        write('If e match ie found, PROLOG returns e "yee".  Otherwiee, it returns a "no".'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('QUESTIONS - continued'),nl,nl,nl,
        writa('Suppose the current databasa conteins the following FACTS:'),nl,nl,
        writa(' vegeteble(squash,yellow).  '),nl,
        write(' vagetabla(radish,red).  '),nl,
        write(' vegeteble(cukes).  '),nl,nl,
        write('Assuming certain user inpute (thosa undarlined), PROLOG will reect es follows:'),nl,nl,
        writa('         | ?- vegetable(aquash).   (RETURN)'),nl,
        write('              ______________________________'),nl,
        write('         no '),nl,
        writa('         | ?- vegetable(equeeh,yellow).  (RETURN)'),nl,
        write('              ____________________________________'),nl,
        write('         yee '),nl,
        writa('         | ?- vegetable(radieh,white).  (RETURN)'),nl,
        write('              ___________________________________'),nl,
        write('         no '),nl,nl,nl,
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Allows the user to repeat any previously presented topic in the tutorial. */

repeatt:- newpg,
          write('After the topic you choose is repeated, TUTOR will continue '),nl,
          write('from where you left off.  These are the topics preceding the '),nl,
          write('lesson you are currently working on:'),nl,nl,
          c_topic(X),
          review_menu(1,X),nl,
          write('Which topic do you want to repeat? (ie "2.  {RETURN}")'),nl,nl,
          mid_input(T2),
          (integer(T2) ->  get_it(T2); newpg, bad_inp, repeatt).

review_menu(B,C):- B == C.
review_menu(B,C):- line(B),
                   B1 is B + 1,
                   review_menu(B1,C).

line(1):- write('1) History of PROLOG').
line(2):- tab(20), write('2) C-PROLOG at KSU'),nl.
line(3):- write('3) facts             ').
line(4):- tab(20), write('4) questions'),nl.
line(5):- write('5) variables         ').
line(6):- tab(20), write('6) conjunctions'),nl.
line(7):- write('7) rules             ').
line(8):- tab(20), write('8) syntax'),nl.
line(9):- write('9) characters        ').
line(10):- tab(20), write('10) operators'),nl.
line(11):- write('11) equality        ').
line(12):- tab(20), write('12) arithmetic'),nl.
line(13):- write('13) lists              ').
line(14):- tab(20), write('14) backtracking'),nl.
line(15):- write('15) cut                ').
line(16):- tab(20), write('16) reading/writing terms'),nl.
line(17):- write('17) reading/writing characters ').
line(18):- tab(9), write('18) accessing files'),nl.
line(19):- write('19) built-in predicates '),nl,nl,nl.
```

```
/* Lesson 7 - RULES */

rules:- newpg,
        write('Lesson 7 - RULES'),nl,nl,
        write('Gerry is an avid reader and reads all types of books.'),nl,nl,
        write('We could define facts like        reads(gerry,novels). '),nl,
        write('                                  reads(gerry,mysteries). '),nl,
        write('                                  reads(gerry,classics).   '),nl,nl,
        write('in PROLOG for every type of book he reads.'),nl,nl,
        write('An easier approach would be to write a RULE like "Gerry reads anything provided'),nl,nl,
        write('it is a book."  Anytime one fact depends on other facts, a RULE is written.'),nl,nl,
        write('The first fact is true IF all those it epends on are true.  A RULE '),nl,nl,
        write('describing Gerrys hobby, "reads(gerry,X):- book(X)."  could be put in'),nl,nl,
        write('the database.   To find out which books Gerry reads, you could now ask the'),nl,nl,
        write('question, "?- reads(gerry,X)"  and the rule would be used to find the answers.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('RULES - continued'),nl,nl,
        write('Definitions are also expressed using RULES.  For example,'),nl,nl,
        write('      X is the daughter of Y and Z if: '),nl,
        write('            X is a female, '),nl,
        write('            Y is the parent of X and '),nl,
        write('            Z is the parent of X. '),nl,nl,
        write('In PROLOG, this RULE would be written,'),nl,nl,
        write('      daughter(X,Y,Z) :- female(X),'),nl,
        write('                         parent_of(Y,X),'),nl,
        write('                         parent_of(Z,X).'),nl,nl,
        write('You could then ask a question like, "?- daughter(joan,X,Y)." to find out'),nl,nl,
        write('which X and Y have a daughter named Joan.  Every X is instantiated to "joan".'),nl,nl,
        write('and each goal to the right of the ":-" is searched for in the database.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('RULES - continued'),nl,nl,
        write('Every RULE consists of a "head" and a "body".  The head contains the'),nl,nl,
        write('overall fact the RULE is trying to prove.  It is never satisfied because it'),nl,nl,
        write('is not a goal.  If the predicate and arguments are matched PROLOG proceeds'),nl,nl,
        write('into the body of clauses.  The body describes the conjunction of goals'),nl,nl,
        write('each separated by a comma, that must be satisfied to prove the head is true.'),nl,nl,
        write('The head and body are always separated by a ":-", which means IF, and'),nl,nl,
        write('each comma in the body a an AND.  As with all other inputs to PROLOG,'),nl,nl,
        write('a rule always end with a period.'),nl,nl,nl,nl,nl,
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Lesson 17 - READING/WRITING CHARACTERS */

rwchar:- newpg,
        write('Lesson 17 - READING/WRITING CHARACTERS'),nl,nl,nl,
        write('The "get0(X)" and "get(X)" goals are used to read a character from the'),nl,nl,
        write('input stream.  If X is uninstantiated, they will always succeed, but like'),nl,nl,
        write('all other input/output operators, they cannot be resatisfied.'),nl,nl,
        write('The first goal "get0(X)" will instantiate X to the next character it sees.'),nl,nl,
        write('It will pick up any printing or non-printing character that is typed in'),nl,nl,
        write('while it is waiting for input.  In the case where X is already instantiated,'),nl,nl,
        write('it checks to see if the next character is equal to X.  If they are equal,'),nl,nl,
        write('"get0(X)" passes, otherwise it fails.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('READING/WRITING CHARACTERS - continued'),nl,nl,nl,
        write('Non-printing characters like space (32 in ASCII) and carriage return'),nl,nl,
        write('are skipped by "get(X)".  It will instantiate X to the first printing'),nl,nl,
        write('character it sees. If X is already instantiated, "get(X)" will compare'),nl,nl,
        write('the next printing character it sees to the value of X.  It tests the two'),nl,nl,
        write('characters for equality and succeeds or fails appropriately.'),nl,nl,nl,
        write('To write out one character, you would use the "put" predicate. '),nl,nl,
        write('X must be instantiated to the ASCII equivalent of the character, and '),nl,nl,
        write('the goal "put(X)" will print the character out.  (ASCII codes are'),nl,nl,
        write('covered in Lesson #9 - CHARACTERS)'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('READING/WRITING CHARACTERS - continued'),nl,nl,nl,
        write('The predicate "put" will always succeed the first time, but cannot be '),nl,nl,
        write('resatisfied. Backtracking over a "put" will fail, but as a side-affect '),nl,nl,
        write('it will output its character again.'),nl,nl,nl,
        write('To control the format of your output, PROLOG has the predicates "nl" and'),nl,nl,
        write('"tab".  A newline "nl" goal will always succeed and move the cursor to'),nl,nl,
        write('the next line.  The "tab(X)" goal will print out X number of spaces.  X must'),nl,nl,
        write('be instantiated to an integer or the goal will fail.'),nl,nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('READING/WRITING CHARACTERS - continued'),nl,nl,nl,
        write('There is one other PROLOG predicate involved in character input'),nl,nl,
        write('called "skip". The goal "skip(M)" will skip to the character right after'),nl,nl,
        write('the next ASCII character M it sees.  M can be any ASCII character or an'),nl,nl,
        write('integer expression.  If the character M is not found and "skip(M)" goes'),nl,nl,
        write('past the end-of-file marker (CTRL-Z or 26 in ASCII), an error '),nl,nl,
        write('will occur. '),nl,nl,nl,nl,
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Lesson 16 - READING/WRITING TERMS */

rwterm:- newpg,
         write('Lesson 16 - READING/WRITING TERMS'),nl,nl,nl,
         write('If you want your program to read in the next term from the current'),nl,nl,
         write('input stream, which is usually the terminal, you would use the "read"'),nl,nl,
         write('predicate.  The term must end with a period and a RETURN or space.'),nl,nl,
         write('Assuming the variable "Reply" is uninstantiated, "read(Reply)" will'),nl,nl,
         write('read in the next term.  "Reply" will be instantiated to everything'),nl,nl,
         write('typed in up until the period and carriage return. '),nl,nl,nl,
         write('The predicate "read" only succeeds once.  It will be skipped when '),nl,nl,
         write('backtracking occurs.'),nl,nl,nl,
         next2,
         mid_input(_),
         newpg,
         write('READING/WRITING TERMS - continued'),nl,nl,nl,nl,
         write('One of the easiest ways to display a term to the current output stream'),nl,nl,
         write('is to use the "write" predicate.  Like "read", "write" succeeds once.'),nl,nl,
         write('The title on this screen is a result of the goal'),nl,nl,
         write('        write('), put(39), write('READING/WRITING TERMS - continued'),put(39),put(41),nl,nl,
         write('The argument of "write" can also be a variable.  If the variable,"X", is'),nl,nl,
         write('instantiated prior to the "write(X)" goal, its value will be dispalyed.'),nl,nl,
         write('Otherwise, a numbered variable like "_375" will be printed.  This is'),nl,nl,
         write('the internal representation for the variable "X" in this instance.'),nl,nl,nl,
         next2,
         mid_input(_),
         newpg,
         write('READING/WRITING TERMS - continued'),nl,nl,nl,
         write('The "write" predicate knows what operator declarations have been made'),nl,nl,
         write('before it prints a term.  The goal '),nl,nl,
         write('                    write(a+b*c)'),nl,nl,
         write('it will print the string "a+b*c" exactly on the screen.  The "+" and'),nl,nl,
         write('"*" are infix operators, so their arguments will be output around them'),nl,nl,
         write('just as they were given to "write".'),nl,nl,nl,nl,nl,
         (c_level(a) -> next2; next3),
         end_input.
```

```
/* Summary 18 - ACCESSING FILES */

s_accfile:- newpg,
          write('Summary 18 - ACCESSING FILES'),nl,nl,nl,
          write('A file called "user" is the initial input and output stream of Prolog.'),nl,nl,
          write('To begin reading input from file "abc", use "see(abc)".'),nl,nl,
          write('The goal "seen" will close the current input file and switch the input'),nl,
          write('stream back to "user".'),nl,nl,
          write('If you want to find out what the current input stream is, use "seeing(W)."'),nl,
          write('"W" will be set to the current input file name.'),nl,nl,
          write('The output stream is handled the same way, with the predicates "tell",'),nl,
          write('"told" and  "telling", respectively.'),nl,nl,
          write('A predicate "consult" will read the contents of a file into the database.'),nl,
          write('The convention "[files]" is often used to consult "files".'),nl,nl,nl,nl,
          (c_level(a) -> next2 ; next3),
          end_input.
```

```
/* Summary 12 - ARITHMETIC */

s_arith:- newpg,
        write('Summary 12 - ARITHMETIC'),nl,nl,nl,
        write('Prolog has 6 infix operators to compare numbers.  They are:'),nl,
        write(' "=" , "\=" , "<"  , ">"  , "=<"   , and  ">=" .'),nl,nl,nl,
        write('The calculation operators are "+", "-", "*" and "/" . '),nl,nl,
        write('A calculation will only occur if the goal contains '),nl,
        write('the "is" operator.'),nl,nl,
        write('An "is" must have a variable on its left, and an expression '),nl,
        write('containing a calculation operator on its right.'),nl,nl,
        write('Divide "/" does integer division.  Its result is always '),nl,
        write('a whole number.'),nl,nl,
        write('The "mod" operator returns only the remainder of an '),nl,
        write('integer division.'),nl,nl,nl,
        (c_level(a) -> next2 ; next3),
        end_input.
```

```
/* Summary 14 - BACKTRACKING */

s_backtr:- newpg,
         write('Summary 14 - BACKTRACKING'),nl,nl,nl,
         write('Backtracking happens automatically, if a goal fails.'),nl,nl,
         write('If a goal fails, Prolog "goes back" to the previous goal.'),nl,nl,
         write('Variables lose their value if backtracked over.  Prolog tries to'),nl,
         write('resatisfy them.'),nl,nl,
         write('The placemarker is moved each time an attempt is made to resatisfy'),nl,
         write('a goal.'),nl,nl,
         write('To force Prolog to backtrack, type "; {RETURN} after Prolog'),nl,
         write('outputs an answer.'),nl,nl,nl,nl,nl,
         (c_level(a) -> next2 ; next3),
         end_input.
```

```
/* Summary 9 - CHARACTERS */

s_chars:- newpg,
          write('Summary 9 - CHARACTERS'),nl,nl,nl,
          write('Prolog has printing and non-printing characters.'),nl,nl,
          write('Each has an ASCII (integer) value.'),nl,nl,
          write('Non-printing characters range from 0 to 32, but their'),nl,
          write('values are machine dependent.'),nl,nl,
          write('Printing characters have values between 33 and 127.'),nl,nl,
          write('Many characters, like the exclamation mark, have a special'),nl,
          write('meaning in Prolog.'),nl,nl,
          write('ASCII values are also used to check character input.'),nl,nl,nl,nl,nl,
          (c_level(a) -> next2 ; next3),
          end_input.
```

```
/* Summary 6 - CONJUNCTIONS */

s_conj:- newpg,
         write('Summary 6 - CONJUNCTIONS'),nl,nl,nl,nl,
         write('A CONJUNCTION combines multiple goals.'),nl,nl,nl,
         write('Each goal is joined by a comma, which means AND.'),nl,nl,nl,
         write('Goals are satisfied from left to right.'),nl,nl,nl,
         write('A variable is SHARED by every goal in a CONJUNCTION.'),nl,nl,nl,
         write('A placemarker is kept in the database for each goal.'),nl,nl,nl,
         write('CONJUNCTIONS can be resatisfied in the same way as a question.'),nl,nl,nl,nl,
         (c_level(a) -> next2; next3),
         end_input.
```

```
/* Summary 15 - CUT */

s_cutt:- newpg,
         write('Summary 15 - CUT'),nl,nl,nl,
         write('There is a goal called CUT.'),nl,nl,
         write('Its predicate is "!", the exclamation point.'),nl,nl,
         write('CUT has no arguments and always succeeds once.'),nl,nl,
         write('It is used to prevent backtracking.'),nl,nl,
         write('Backtracking over a CUT always fails.'),nl,nl,
         write('A placemarker is NOT kept for any goal that preceeds a CUT.'),nl,
         write('They can never be resatisfied.'),nl,nl,nl,nl,
         (c_level(a) -> next2; next3),
         end_input.
```

```
/* Summary 11 - EQUALITY */

s_equal:- newpg,
          write('Summary 11 - EQUALITY'),nl,nl,nl,
          write('The infix operator "=" means equality in Prolog.'),nl,nl,
          write('An integer or atom is always equal to itself.'),nl,nl,
          write('When two uninstantiated variables are equal, they'),nl,
          write('will share the same value.'),nl,nl,
          write('A variable set equal to an object will assume the value '),nl,
          write('of the object.'),nl,nl,
          write('Two structures are EQUAL if:'),nl,
          write('         - they both have the same predicate  AND'),nl,
          write('         - they both have the same number of arguments  AND'),nl,
          write('         - each corresponding argument is equal.'),nl,nl,
          write('The opposite of "=" is "/=".'),nl,nl,nl,
          (c_level(a) -> next2; next3),
          end_input.



/* Summary 3 - FACTS */

s_facts:-newpg,
         write('Summary 3 - FACTS'),nl,nl,nl,
         write(' The relationship is always written first.'),nl,nl,
         write(' Names of relationships and objects must begin with a'),nl,
         write(' lower-case letter.'),nl,nl,
         write(' The objects are enclosed in a set of parentheses and'),nl,
         write(' separated by commas.'),nl,nl,
         write(' Each FACT ends with a period.'),nl,nl,
         write(' Every object within the parentheses is called an ARGUMENT.'),nl,nl,
         write(' The relationship between objects is called a PREDICATE.'),nl,nl,
         write(' A collection of FACTS is called a DATABASE.'),nl,nl,nl,nl,nl,
         (c_level(e) -> next2; next3),
         end_input.



/* Summary 13 - LISTS */

s_lists:- newpg,
          write('Summary 13 - LISTS'),nl,nl,nl,
          write('A LIST is an ordered sequence of elements.  It can be of any length.'),nl,nl,
          write('The elements are enclosed in square brackets and separated by commas.'),nl,nl,
          write('An empty list is written "[]".'),nl,nl,
          write('The elements can be atoms, structures or other lists.'),nl,nl,
          write('The HEAD of a list is the first element.'),nl,nl,
          write('The TAIL is the list without its HEAD.'),nl,nl,
          write('Be careful not to confuse the HEAD of a list with the HEAD'),nl,
          write('of a rule.  The two are totally different.'),nl,nl,nl,nl,
          (c_level(a) -> next2; next3),
          end_input.
```

```
/* Summary 10 - OPERATORS */

s_oper:- newpg,
         write('Summary 10 - OPERATORS'),nl,nl,nl,
         write('An INFIX operator is written between its arguments.'),nl,nl,nl,
         write('PREFIX operators come before their argument.'),nl,nl,nl,
         write('An operator written after its argument is called POSTFIX.'),nl,nl,nl,
         write('The operator with the highest PRECEDENCE is evaluated first.'),nl,nl,nl,
         write('Operators with the same precedence are evaluated left-to-right.'),nl,nl,nl,nl,nl,
         (c_level(s) -> next2; next3),
         end_input.



/* Summary 4 - QUESTIONS */

s_quest:-newpg,
         write('Summary 4 - QUESTIONS'),nl,nl,nl,
         write('QUESTIONS always begin with the 2 symbols " ?- ".'),nl,nl,
         write('The format of a QUESTION is very similar to that of a FACT.'),nl,nl,
         write('A QUESTION always ends with a period.'),nl,nl,
         write('When the PROLOG interpreter receives a QUESTION, it searches the current'),nl,
         write('database for a possible match.  If a match is found, it returns a "yes".'),nl,
         write('If no match is found, PROLOG returns a "no". '),nl,nl,
         write('QUESTIONS are not kept in PROLOG"s database like FACTS.'),nl,nl,
         write('DO NOT confuse "?-" with the symbols "| ?-".  The latter set of symbols'),nl,
         write('will appear when the PROLOG interpreter is waiting for a command.'),nl,nl,nl,nl,
         (c_level(s) -> next2; next3),
         end_input.



/* Summary 7 - RULES */

s_rules:- newpg,
          write('Summary 7 - RULES'),nl,nl,nl,
          write('A RULE defines a FACT that depends on other FACTS.'),nl,nl,
          write('RULES are stored in the database, like FACTS.'),nl,nl,
          write('The HEAD of a RULE defines a general FACT.'),nl,nl,
          write('The BODY is made up of goals that must all be proven'),nl,
          write('before the HEAD is true.'),nl,nl,
          write('To separate the HEAD from the BODY, the symbols ":-" are used.'),nl,nl,nl,nl,
          (c_level(s) -> next2; next3),
          end_input.



/* Summary 17 - READING/WRITING CHARACTERS */

s_rwchar:- newpg,
           write('Summary 17 - READING/WRITING CHARACTERS'),nl,nl,nl,
           write('If X is uninstantiated:'),nl,nl,
           write('      "get0(X)" reads in the next printing or non-printing character.'),nl,nl,
           write('      "get(X)" reads in the next printing character.'),nl,nl,
           write('If X is instantiated:'),nl,nl,
           write('      "get0(X)" checks to see if the next character equals X.'),nl,
           write('      If so, it succeeds.'),nl,nl,
           write('      "get(X)" checks the next printing character.'),nl,nl,
           write('The "put" predicate will output one character at a time.'),nl,nl,
           write('Spaces are output using "tab" and a newline is done with "nl".'),nl,nl,nl,
           (c_level(s) -> next2; next3),
           end_input.
```

```
/* Summary 16 - READING/WRITING TERMS */

a_rwterm:- newpg,
           write('Summary 16 - READING/WRITING TERMS'),nl,nl,nl,
           write('The predicate "read" will read in a term from the current'),nl,
           write('input stream.'),nl,nl,
           write('The TERM must end with a period and a RETURN or a space.'),nl,nl,nl,
           write('"read" only succeeds once.'),nl,nl,nl,
           write('To output a TERM you can use the "write" predicate.'),nl,nl,
           write('Its argument must be enclosed in single quotes if it'),nl,
           write('is a string.'),nl,nl,nl,nl,
           (c_level(e) -> next2; next3),
           end_input.




/* Summary 8 - SYNTAX */

a_syntax:- newpg,
           write('Summary 8 - SYNTAX'),nl,nl,nl,
           write('Prolog checks each character of input for syntax errors.'),nl,nl,
           write('A Prolog program is a set of TERMS.'),nl,nl,
           write('Terms are CONSTANTS, VARIABLES or STRUCTURES made up of CHARACTERS.'),nl,nl,
           write('An ATOM is a CONSTANT and '),nl,
           write('    - starts with a lower-case letter and contains'),nl,
           write('      lower-case letters and digits.  OR'),nl,
           write('    - is made up of all symbols.  OR'),nl,
           write('    - contains any letter, but is enclosed in single quotes.'),nl,nl,
           write('An INTEGER is a CONSTANT and must be a whole number.'),nl,nl,
           write('VARIABLES begin with a capital letter or an underscore.'),nl,nl,
           write('A STRUCTURE is made up of a predicate and arguments.'),nl,nl,nl,
           (c_level(a) -> next2; next3),
           end_input.




/* Summary 5 - VARIABLES */

a_vars:-newpg,
        write('Summary 5 - VARIABLES'),nl,nl,nl,
        write('A VARIABLE always begins with a capital letter.'),nl,nl,nl,
        write('VARIABLES always start out UNINSTANTIATED, in other words,'),nl,
        write('without a value.'),nl,nl,nl,
        write('When a match is found, the VARIABLE becomes INSTANTIATED, or bound'),nl,
        write('to the value of the argument it matched.'),nl,nl,nl,
        write('The predicate of a clause cannot be a VARIABLE.  For example, '),nl,
        write('XYZ(a, b, c). is a syntax error because XYZ is a VARIABLE.'),nl,nl,nl,nl,
        (c_level(a) -> next2; next3),
        end_input.
```

```
/* Welcome screen to the TUTOR tool.  First it asserts the maximum topic
   number (+1) so it can control the beginner and beginner/intermediate mode  */

start:- asserta(max(20)),
        newpg,
        write('               Welcome to TUTOR -- a C-PROLOG tutorial'),nl,nl,
        write('This tool will teach you C-Prolog.  It will guide you through the language,'),nl,
        write('according to your skill level, by presenting topics and then asking questions.'),nl,nl,
        write('The following topics are covered:'),nl,nl,
        menu,
        write('It is strongly suggested that you obtain the "TUTOR Users Guide" from'),nl,nl,
        write('the Computer Science Department before proceeding with the tutorial.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('WHENEVER you type in a response, ALWAYS end it with a period and a '),nl,nl,
        write('carriage return {RETURN}.  This way, Prolog will know when you are done.'),nl,nl,nl,
        write('You can leave TUTOR at any time by typing  "halt."  "q."  "quit." or "exit."'),nl,nl,
        write('What is your skill level ?'),nl,nl,
        write(' Beginner - Type "b.  {RETURN}"'),nl,
        write('         Assumes no prior knowledge of PROLOG - automatically'),nl,
        write('         guides you through every topic.'),nl,nl,
        write(' Intermediate - Type "i.  {RETURN}"'),nl,
        write('         Allows you to resume from where you left off during'),nl,
        write('         last session, or to choose one topic at a time.'),nl,nl,
        write(' Experienced - Type "e.  {RETURN}"'),nl,
        write('         Allows you to pick one particular topic at a time.'),nl,nl,
        next1,
        mid_input(Sk1),
        level(Sk1).

level(i):- asserta(c_level(i)), topic('/usr/prolog/interm', interm, 0).

level(e):- asserta(c_level(e)), topic('/usr/prolog/exper', exper, 0).

level(_):- max(M), asserta(c_level(b)), seq(1,M), halt.
```

```
/* Lesson 8 - SYNTAX */

syntax:- newpg,
        write('Lesson 8 - SYNTAX'),nl,nl,nl,
        write('In order to correctly represent data in PROLOG or any other language,'),nl,nl,
        write('the SYNTAX rules must be followed.  As PROLOG reads in each CHARACTER,'),nl,nl,
        write('it is constantly checking the SYNTAX rules.'),nl,nl,nl,
        write('A program in PROLOG is really a set of TERMS.  A TERM is a CONSTANT,'),nl,nl,
        write('VARIABLE or a STRUCTURE.  Each TERM is written as a sequence of CHARACTERS.'),nl,nl,
        write('A CHARACTER is either UPPER-CASE, LOWER-CASE, a DIGIT or a SYMBOL.'),nl,nl,
        write('There are SYNTAX rules defined for how each type of TERM uses CHARACTERS'),nl,nl,
        write('to form names.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('SYNTAX - continued'),nl,nl,nl,
        write('A specific object or relationship is represented by a CONSTANT.'),nl,nl,
        write('CONSTANTS are divided into two categories: ATOMS and INTEGERS.'),nl,nl,
        write('Earlier lessons were full of atoms like:'),nl,nl,
        write('    reeds, john, fish, rosemary, music, soccer, dangerous'),nl,nl,nl,
        write('Symbols like "?-" and ":-" are also atoms.  Normally, an atom consists'),nl,nl,
        write('of lower-case letters and digits and begins with a lower-case letter, OR'),nl,nl,
        write('is made up of all symbols.  To define an atom that combines these or'),nl,nl,
        write('uses a capital letter, it must be enclosed in single quotes.'),nl,nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('SYNTAX - continued'),nl,nl,nl,
        write('The other type of CONSTANT is an INTEGER.  They are mostly used to '),nl,nl,
        write('represent numbers in arithmetic operations.  An INTEGER is a whole number'),nl,nl,
        write('and must not contain a decimal point.'),nl,nl,nl,
        write('VARIABLES are like ATOMS but begin with a capital letter or the underscore'),nl,nl,
        write('"_" symbol.  They usually represent unknown objects.  The name of a'),nl,nl,
        write('VARIABLE can be practically any length.  The ANONYMOUS VARIABLE, "_", '),nl,nl,
        write('is used, for example, when we want to know if John plays a sport but it'),nl,nl,
        write('does not matter what the sport is.  The question "?- plays(john,_)." will'),nl,nl,
        write('return a "yes" or "no" instead of the variable and its value.'),nl,nl,
        next2,
        mid_input(_),
        newpg,
        write('SYNTAX - continued'),nl,nl,nl,
        write('The third kind of TERM is the STRUCTURE.  A STRUCTURE consists of a'),nl,nl,
        write('predicate, which is an ATOM, and its arguments, which are TERMS.  The'),nl,nl,
        write('arguments are enclosed in parentheses and separated by commas.'),nl,nl,
        write('These are all STRUCTURES: '),nl,nl,
        write('    owns(mary, book( a_tale_of_two_cities, dickens)).'),nl,nl,
        write('    looks(john, tired).'),nl,nl,
        write('    ?- read(mary, plays(X, shakespeare)).'),nl,nl,nl,
        write('Anything that is not a CONSTANT or VARIABLE, by default, is a STRUCTURE.'),nl,nl,nl,nl,
        (c_level(e) -> next2; next3),
        end_input.
```

```
/* Main file that consults all necessary files */

:-([
        '/uar/prolog/lib',              % library
        '/usr/prolog/start',
        '/uar/prolog/ask',
        '/usr/prolog/mult',
        '/usr/prolog/rapeatt',
        '/usr/prolog/get_it'
  ]).
:-nl.
:-write('*******************************************************'),nl.
:-write('***      To begin, type   start. {RETURN}      ***'),nl.
:-write('***                       ---------------      ***'),nl.
:-write('***                                            ***'),nl.
:-write('***      To quit,  type   halt. {RETURN}       ***'),nl.
:-write('***                       --------------       ***'),nl.
:-write('*******************************************************'),nl.
:-nl.
```

```
/* Lesson 5 - VARIABLES */

vars:- newpg,
       write('Lesson 5 - VARIABLES'),nl,nl,nl,
       write('VARIABLES begin with an upper-case letter or an underscore.  The arguments of'),nl,nl,
       write('a QUESTION can be VARIABLES as well as constants.  With a database like,'),nl,nl,
       write('  equipment(helmet,football).'),nl,
       write('  equipment(javelin,track).'),nl,
       write('  equipment(discus,track).'),nl,nl,
       write('you could ask PROLOG,  "Is a helmet track equipment?",  "Is a javelin track'),nl,nl,
       write('equipment?" and " Is a discus track equipment?"  to figure out which items'),nl,nl,
       write('in the database were track equipment.'),nl,nl,nl,
       write('Using a VARIABLE makes this job a lot easier.  You can ask one question,'),nl,nl,
       write('"Which items are track equipment?"  or  "?-equipment(X,track)."'),nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('VARIABLES - continued'),nl,nl,nl,
       write('With that QUESTION, PROLOG would start a top-down search of the current'),nl,nl,
       write('database to find every FACT with 2 arguments, whose functor is "equipment", '),nl,nl,
       write('and second argument is "track". Until a match is found, X is UNINSTANTIATED,'),nl,nl,
       write('meaning, it has no value. '),nl,nl,
       write('If and when a match is found, X becomes INSTANTIATED to the value of the'),nl,nl,
       write('argument it matched, PROLOG marks the location in the database where the '),nl,nl,
       write('match was found and it prints out the VARIABLE name and its value.'),nl,nl,
       write('At this point, you can have PROLOG continue to search the database for more'),nl,nl,
       write('matches or stop the search.  PROLOG will wait for further instructions.'),nl,nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('VARIABLES - continued'),nl,nl,nl,
       write('To terminate the search, you have to press the RETURN key.  To continue'),nl,nl,
       write('the search, type a semicolon and then the RETURN key.  PROLOG forces X to'),nl,nl,
       write('become UNINSTANTIATED again and resumes its search from the place it marked'),nl,nl,
       write('in the database.  If no more matches are found, PROLOG returns a "no".'),nl,nl,nl,nl,
       next2,
       mid_input(_),
       newpg,
       write('VARIABLES - continued'),nl,nl,
       write('Given the current database, if the user types each item underlined, he will'),nl,nl,
       write('see the following:'),nl,nl,nl,
       write('  CURRENT DATABASE                     PROLOG INTERPRETER'),nl,nl,
       write(' vegetable(lettuce,green).           | ?- vegetable(X,green).  {RETURN}'),nl,
       write('                                          -----------------------------'),nl,
       write(' vegetable(beans,green).             X = lettuce ;  { RETURN }'),nl,
       write('                                                   -------------'),nl,
       write(' vegetable(radish,red).              X = beans ;    { RETURN }'),nl,
       write('                                                 ---------------'),nl,
       write(' vegetable(peppers,green).           X = peppers ;  { RETURN }'),nl,
       write('                                                   -------------'),nl,
       write('                                     no '),nl,nl,nl,
       (c_level(a) -> next2; next3),
       end_input.
```

TUTOR : A Computer-Aided Tutorial in PROLOG

BY

LISA MARIE WYLIE

B. S., Worcester Polytechnic Institute, 1980

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

ABSTRACT

Prolog is an interpretive language for PROgramming in LOGic. A program written in Prolog contains facts and rules which describe characteristics of an application. These are stored in the Prolog database. To solve a problem, the database is queried.

This work presents an interactive, instructional tool called TUTOR that teaches a version of Prolog called C-Prolog. It is written in C-Prolog and is available on the VAX 11/780, Perkin Elmer 8/32 and the PLEXUS systems at Kansas State University. Code size of the tutorial is approximately 80K bytes.

TUTOR accommodates users of three skill levels. The history of Prolog and an overview of C-Prolog are described in the first two lessons. The major areas of the language are covered in the next sixteen lesson/summary/exercise sections. The last lesson briefly describes the core set of builtin predicates of C-Prolog.