

DTIC FILE COPY

2

AD-A201 028

NAVAL POSTGRADUATE SCHOOL Monterey, California



THESIS

DTIC
ELECTE
NOV 15 1988
S D
CH

A COMPUTER-AIDED INSTRUCTION
PROGRAM FOR TEACHING
THE TOPS20-MM FACILITY ON THE DDN

by

Tae Woo Kim

June 1988

Thesis Advisor:

Neil C. Rowe

Approved for public release; distribution is unlimited

88 11 15 008

REPORT DOCUMENTATION PAGE AD A 2-1028					
1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS			
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution is unlimited			
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5. MONITORING ORGANIZATION REPORT NUMBER(S)			
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL (If applicable) Code 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School			
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000		7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000			
8a. NAME OF FUNDING / SPONSORING ORGANIZATION	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS (City, State, and ZIP Code)					
		10. SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) A COMPUTER-AIDED INSTRUCTION PROGRAM FOR TEACHING THE TOPS20-MM FACILITY ON THE DDN					
12. PERSONAL AUTHOR(S) Kim, Tae Woo					
13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 1988 June	15. PAGE COUNT 54		
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Computer Assisted Instruction; Artificial Intelligence			
FIELD	GROUP				SUB-GROUP
19. ABSTRACT (Continue on reverse if necessary and identify by block number) We constructed an Intelligent Computer Assisted Instruction (ICAI) program to tutor the usage of the TOPS20-MM, an electronic mail facility available on the Defense Data Network (DDN). Learning by experience is one of the best ways to learn something. The main strategy of tutoring in this thesis was to provide an environment simulating the actual facility that guides the student while he/she tries to perform given tasks. Means-ends analysis, a classic technique for solving search problems in Artificial Intelligence, has been used to figure out the right command to perform a given task. Basic commands, e.g. a command for viewing a message number 7, will be taught first, then the tasks like "Send a message to the people mentioned in message number 3" will be issued. <i>Thesis</i> <i>(SDU)</i>					
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION Unclassified			
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. Neil C. Rowe	22b. TELEPHONE (Include Area Code) (408) 646-2462	22c. OFFICE SYMBOL Code 52Rp			

Approved for public release; distribution is unlimited.

**A COMPUTER-AIDED INSTRUCTION
PROGRAM FOR TEACHING
THE TOPS20-MM FACILITY ON THE DDN**

by

**Tae Woo Kim
Capt. Republic of Korea Army
B.S., Korea Military Academy, 1984**

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1988

Author:

김 태 우

Tae Woo Kim

Approved by:

Neil C. Rowe

Neil C. Rowe, Thesis Advisor

Gary K. Poock

Gary K. Poock, Second Reader

Michael D. Zyda

For Robert B. McGhee, Acting Chairman,
Department of Computer Science

James M. Fremgen

James M. Fremgen,
Acting Dean of Information and Policy Sciences

ABSTRACT

We constructed an Intelligent Computer Assisted Instruction (ICAI) program to tutor the usage of the TOPS20-MM, an electronic mail facility available on the Defense Data Network (DDN). Learning by experience is one of the best ways to learn something. The main strategy of tutoring in this thesis was to provide an environment simulating the actual facility that guides the student while he/she tries to perform given tasks. Means-ends analysis, a classic technique for solving search problems in Artificial Intelligence, has been used to figure out the right command to perform a given task. Basic commands, e.g. a command for viewing a message number 7, will be taught first, then the tasks like "Send a message to the people mentioned in message number 3" will be issued.



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I. INTRODUCTION	1
II. SURVEY OF PREVIOUS WORK	3
A. INTELLIGENT TUTORING SYSTEMS	3
1. Overview	3
2. Intelligent CAI Systems	4
a. SOPHIE	4
b. GUIDON	4
c. EXCHECK	5
d. The C Workshop	6
III. Description of the Application	7
A. About DDN	7
B. About TOPS20-MM	7
IV. Implementation of the Tutor	11
A. Data Structures and the System Environment	11
B. Structure of the program	11
1. Problem Domain Module	11
2. Tutor Module	12
3. TOPS20-MM Module	13
V. CONCLUSIONS	15
APPENDIX A - DEMONSTRATION	16
APPENDIX B - SOURCE PROGRAMS	21
LIST OF REFERENCES	46
INITIAL DISTRIBUTION LIST	47

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my thesis advisor, Professor Neil C. Rowe, for his ceaseless help and guidance throughout the thesis and my second reader, Professor Gary K. Poock, who taught me all about the DDN and the TOPS20-MM.

Now, it is time to go home and I'd like to thank my family, far away in Korea yet stayed within my heart all the way through....

I. INTRODUCTION

This thesis built a tutor to teach an electronic mail facility called "TOPS20-MM". The computer serves as a nice tool to do this. *Learning by experience* is one of the best ways to learn something. This thesis provides a simulation of the MM facility environment and the student learns by trying to perform given tasks with it. TOPS20-MM, which we have chosen to teach, is the most-used service on the Defense Data Network(DDN). The DDN and the TOPS20-MM are discussed briefly in chapter 3.

Computer-Assisted Instruction(CAI) programs try to encourage and control learning while the student is involved in some activity. Artificial intelligence in computer-based instructional applications provides a new kind of learning environment. Some of the important applications of artificial-intelligence-based teaching are discussed in chapter 2.

The strategy we have used for teaching is: (1) issue a task or an exercise to the student; (2) monitor the student's actions; (3) compare the command issued by the student with the correct command; (4) either perform the actual MM actions or correct the error. For the tutor to best teach the student the commands and procedures to use the MM facility, it should be able to figure them out itself. To do so, we have exploited means-ends analysis, a classic technique for solving search problems by abstraction. Means-ends analysis tries to find appropriate operators to reach a certain goal state from a starting state by using the preconditions and postconditions of each operator. In our program, we have determined preconditions and postconditions of each command.

For the tasks to be issued, we have used a subskill lattice. With this idea, the basic commands available in MM will be taught first, then more complicated tasks which are compositions of basic ones. The tutoring program is written in PROLOG and it runs on UNIX.

II. SURVEY OF PREVIOUS WORK

A. INTELLIGENT TUTORING SYSTEMS

1. Overview

The goal of computer aided instruction (CAI) research is to build instructional programs that incorporate well-prepared course material in lessons that are optimized for each student [Ref. 1]. In the 1970's, course materials were represented independently of teaching procedures by intelligent computer-aided instruction (ICAI) programs, so that problems and tutoring advice could be generated differently for each student. In recent years, researchers have concentrated on supportive learning environments to provide *learning-by-doing*, which means a student acquires knowledge by solving real-world problems [Ref. 2]. To tutor well, a system has to specify exercises and their answers ahead of time, and must have an ability to diagnose or model the student's behavior. It must also have tutorial strategies that specify when to interrupt a student and what to say then.

When CAI researchers realized the complexity of such a computer-based tutor, they began to apply artificial intelligence (AI) techniques. AI research such as natural-language understanding, knowledge representation, and inferencing, have been applied. The main components of an such "intelligent computer-aided instruction" (ICAI) systems are *problem-solving expertise*, what the system tries to teach the student; *the student model*, what the student does and does not know; and *tutoring strategies*, how the system presents material to the student.

2. Intelligent CAI Systems

In this section four intelligent computer-aided instruction (ICAI) systems relevant to this thesis are discussed briefly.

a. SOPHIE

SOPHIE (a SOPHisticated Instructional Environment) is an ICAI system developed by John Seely Brown, Richard Burton, and their colleagues at Bolt Beranek and Newman, Inc [Ref.3:p.247]. It tries to teach natural proofs as they are actually done by mathematicians. It is designed to provide the student with a learning environment in which he/she acquires problem-solving skills by making hypotheses and trying out ideas.

SOPHIE has modules for problem-solving knowledge, heuristic strategies to answer the student's questions, rules to criticize the student's hypotheses, and the rules to suggest alternative theories for the current hypothesis. In a simulated electronics laboratory, it teaches problem-solving skills. It issues problems to the student to find the faults in a malfunctioning electric circuit. After the student measures the equipment and proposes a solution, he/she receives feedback as to the logical validity of their proposed solutions. When the student makes an error in his/her logic, SOPHIE can critique by generating relevant counterexamples. The SOPHIE system combines domain-dependent knowledge and domain-independent inferencing mechanisms to answer questions.

b. GUIDON

GUIDON was developed by William J. Clancey and his colleagues at Stanford University [Ref.3:p.267]. It is design to answer "how the problem-solving rules in the MYCIN consultation system can be used by other tutoring programs in interaction with a student?" and "what can be done to MYCIN in order to make it more effective tutorial program?" MYCIN is a rule-based expert system that diagnoses the cause of the infection using knowledge relating infecting organisms with patient history and test results [Ref.4:p.283].

GUIDON uses the rules of the MYCIN consultation system as subject material and organizes them into procedures. MYCIN's rules are not been modified, but are used to form quizzes, guide the dialogue, summarize evidence, and model the student's understanding for the tutoring application. GUIDON engages the student in a dialogue about a patient suspected of having an infection. The student learns the relevant clinical and laboratory data and how to use it to diagnose the patient.

GUIDON differs from other ICAI programs in its mixed-initiative dialogue, which allows both tutor and student decide what to do next. It uses structured teaching interactions that record the previous dialogue, and it does more than just respond to the student's last action. It demonstrates that by separating teaching knowledge from subject knowledge, we can treat the teaching knowledge as a rule-based system itself.

c. EXCHECK

EXCHECK is an instructional system developed by Patrick Suppes and his colleagues at the Institute for Mathematical Studies in the Social Sciences (IMSSS) at Stanford University [Ref.3:p.283]. It tries to teach natural proofs as they are actually done by mathematicians, by checking student proofs.

EXCHECK has natural-inference subroutines which allow it to understand sketches of proofs, summarize proofs, and explain set theory. These subroutines allow the system to interact in a natural style which is similar to standard mathematical practice. With this system, the student is given lesson materials first, then exercises consisting of theorems are given for him/her to prove. Since its inference procedures allow the system to recognize the student's reasoning and track his/her solutions, EXCHECK has similarities to SOPHIE.

d. The C Workshop

The C workshop is a tutorial program developed by Charles Pine at Word Craft [Ref. 5]. It teaches the C programming language by providing a programming environment and online help facility.

It has a tutoring module called Soft Tutor that examines the workings of the student's program. The system presents example programs, and the student modifies them to obtain his/her own programs. The Soft Tutor gives immediate feedback by error messages. It uses an actual C compiler to teach the C language since its problem domain is a computer system. It provides an working environment, but it lacks much interactive ability.

III. Description of the Application

A. About DDN

The Defense Data Network (DDN) is a large common-user data communications network operated for the Department of Defense (DoD) by the Defense Data Network Program Management Office (DDN PMO) of the Defense Communications Agency (DCA). Mail services let users send messages electronically to one another. They have these features:

1. *Reading messages*: select and view specific messages.
2. *Printing, deleting, or moving messages*: print messages on the printer, move them into files, or delete them.
3. *Sending messages*: send messages to other users; you must know the network mailbox or "address" of the addressee.

In this thesis, I'll concentrate on the "TOPS20 MM" mail service.

B. About TOPS20-MM

MM is a sophisticated program used to create and manipulate messages [Ref. 6]. MM has three levels: the TOPS20 executive level which displays "@" prompt on the screen; the main executive level of MM that displays "MM>" prompt; and the sub-modes that can be entered from MM level.

The top level is the initial and main command level. Here most of MM's power is available through a variety of commands. Available commands and their functions are as follows:

ALIAS: allows you to use MM as the given user.

ANSWER: allows you to respond a selected message individually.

APPEND: concatenates selected messages into one.

BBOARD: allows you to select a file name from a list of names for MM to work.

BLANK: clears the screen.

BUG: allows you to report or send suggestions to the MM maintainer.

CHECK: shows header-lines of newly arrived messages if any.

CONTINUE: resumes the last SEND, if it was aborted.

COPY: copies selected messages to the given file name.

COUNT: gives the number of the required messages.

CREATE-INIT: makes a new version of MM.INIT in your login directory.

DAYTIME: gives you the current date and time.

DELETE: sets the delete bit to the selected messages.

DISABLE: makes the current message into read-only mode.

ECHO: shows a text string you type on the terminal.

EDIT: allows you to edit selected messages with the editor.

ENABLE: makes the current message into read-write mode.

EXAMINE: sets read-only mode to allow you to select a message for MM to work.

EXIT: halts MM after removing all the deleted messages in the mail file.

EXPUNGE: just removes all the deleted messages in the mail file.

FILE-LIST: shows an index of header lines.

FLAG: just sets the flag bit of selected messages.

FORWARD: allows you to send selected messages with your comments.

FROM: allows you to specify the address field with other commands.

GET: allows you to specify the mail file for MM to work with.

HEADERS: shows the header line of messages you selected.

HELP: shows a definition of the following command.

JUMP: changes the current message to be the following number.

LIST: shows an index of header lines and then prints the selected messages.

LITERAL-TYPE: shows the selected messages without regard to the variables only-type-headers and dont-type-headers.

LOGOUT: expunges the mail file, and logs you out of the system.

MARK: sets the seen bit of selected messages.

MOVE: copies selected messages to the end of given file and sets a delete bit to the selected messages.

NET-MAIL: tries to send queued mail, if any.

NEXT: types the next message if undeleted, and makes it the current message.

PREVIOUS: types the previous message if undeleted, and makes it the current message.

PROFILE: allows you to establish values for a basic set of variables to adjust MM's environment to your preference.

PUSH: gives you a fresh EXEC after placing the current job in an inferior fork.

QUIT: halts MM without removing deleted messages in the current mail file.

READ: allows you to enter READ mode.

REMAIL: allows you to resend selected messages without comments.

REPLY: allows you to respond each selected message individually.

REPLY-TO: adds the address to the "Reply-To:" field for all outgoing messages.

RESTORE-DRAFT: enters SEND-mode with resetting the outgoing message fields to the values of the message draft saved in the given file.

SEND: allows you to enter SEND-mode.

SET: allows you to change the value of the selected variable for this session.

SHOW: shows the current MM variable settings.

SORT: reorders the selected messages chronologically by date of composition.

STATUS: tells you relevant information and statistics of the current mail file.

SYSTEM-MSGs: is the same as GET SYSTEM:MAIL.TXT.1

TAKE: allows you to change the input source from the terminal to the file.

TYPE: shows the selected messages, considering the variables only-type-headers, and dont-type-headers and sets the seen bit.

UNANSWER: removes the ANSWER bit from the selected messages.

UNDELETE: removes the DELETE bit from the selected messages.

UNFLAG: removes the FLAG bit from the selected messages.

UNKEYWORDS: removes the given keyflag and keywords from the "Keywords:" field of selected messages.

UNMARK: removes the SEEN bit from selected messages.

VERSION: gives the version number and configuration of MM.

IV. Implementation of the Tutor

A. Data Structures and the System Environment

This program has been developed on VAX 11/785 running the UNIX Operating System. It is written in C-Prolog, a Prolog interpreter written in C for 32 bit machines.

The program uses code for means-ends analysis written by Prof. Rowe [Ref. 1]. Means-ends analysis tries to find a path from a starting state to a goal state, representing it by a sequence of operators. In our program, operators are the available commands in each mode. To reach the goal states, recommended operators are specified and preconditions are specified for each operator. Also, deletepostconditions and addpostconditions are defined for each operators. Lists hold the sequence of operators to be issued to perform a given task. Lists also represent preconditions and postconditions for each operator, the starting state, and the goal state.

B. Structure of the program

The program is divided into three modules. The problem domain module defines the tasks issued to the student to teach the TOPS20-MM facility. The tutor module implements various tutoring strategies. The TOPS20-MM module simulates the actual TOPS20-MM environment.

1. Problem Domain Module

Student tasks are divided into two groups: basic and advanced. Basic tasks teach the concept of domain levels and single basic commands in each level. The commands we chose are a subset of the commands described in previous chapter:

to teach answer, copy, delete, exit, expunge, flag, forward,
headers all, headers answer, headers current, headers delete,

headers flag, headers unseen, help, jump, list, logout, move,
next, previous, quit, read, send, type, undelete, and unflag.

2. Tutor Module

The tutor issues tasks to the student and monitors the actions performed. Most of its code was written by Prof. Rowe. It contains an inference engine for means-ends analysis, tutoring strategies, an interface, a help facility, and utilities. The tutor issues a task, compares the student's answer with the correct command figured out by the inference module, and gives the student appropriate assistance and corrects any errors. The tutor interacts with the student via the interface module and provides help with the help facility module. It executes commands the student types in by invoking the TOPS20-MM module, if it is a correct command.

For example, in order to send a message, a user must type the command "send" to enter the SEND mode from MM or READ mode; "send" is thus an operator. Means-ends analysis requires clear definitions of the preconditions and postconditions for each operator. In this case, the precondition can be expressed by:

precondition(command(mm,send),[level(mm)])

where the second argument is the precondition that the current level must be MM. After an operator has been applied, it must delete previous conditions which are no longer true and add new conditions that became true. These can be represented by:

deletepostcondition(command(mm,send),[level(mm)])

and

addpostcondition(command(mm,send),[level(send),issued_command(mm,send)]),

respectively. The recommended operator to reach a goal state is represented by: predicate "recommended" as in

recommended([level(send)],command(mm,send))

which can be read as "In order to reach the state where the current level is SEND, try to

issue a send command from the MM level." Means-ends analysis figures out the required sequence of operators to accomplish a task by using these conditions.

After issuing the task and receiving a command from the student, the tutor uses some of the teaching strategy rules in metutor9 module written by Prof. Rowe to decide what actions to take according to the student's input, plus some code of my own. If the received command was invalid, the tutor checks for minor errors such as mistyping or transposition of characters in a command. It infers the correct command if it recognizes one difference in character string of the received command from a valid one, if by changing the positions of two adjacent characters in the string; then it will ask the student if it can correct the command.

A subroutine called "niceread" has been written by Prof. Rowe to make input and output process a little easier. The tutor will recognize input when the user hits the return key without an extra period. When the tutor must talk to the student while he/she is inside the environment of the TOPS20-MM, the words are preceded by "TUTOR:"

A help facility can show the available commands at each level and explain the functions of each command, when requested by the student typing "?" or "help". The student can also review the task by typing "task".

3. TOPS20-MM Module

When the student types a correct command, the tutor passes it and the current level to the TOPS20-MM module for execution. For example, after a "send" command, it executes the action rule whose left hand side is action(mm,send) and performs the function of the command. It then enters the "To:" level and waits for the address of the receiver.

Our program differs from the actual TOPS20-MM in the usage of the escape character. MM allows the student to abbreviate a command by typing two or three

characters followed by an escape character, but it was hard to implement and was discarded from this module. Instead, the exact commands were required to execute the system.

V. CONCLUSIONS

We built a computer tutor to teach part of MM. We showed it was possible to provided an efficient working environment for a student to learn the usage of MM by issuing the commands and seeing results. Since the time was limited, only a part of MM was addressed.

"Is it cost effective?" is an important question when developing an ICAI system. TOPS20-MM is the most-used service provided in the Defense Data Network. Since teaching TOPS20-MM isn't complex, a computer tutor can save much time and effort of a teacher as well as a student.

The program didn't cover all of MM, but it demonstrated that artificial intelligence can be added to computer aided-instruction to provide a better system. It could correct typing errors and misconceptions in level, precondition violations, and other more subtle errors. The program could be improved by providing a complete simulation of MM, and more tutoring strategies, and computer graphics.

APPENDIX A - DEMONSTRATION

Tutor: Hi, what is your name? Kim

Tutor: Nice to meet you, Kim.

Welcome to TOPS20 MM Self-learning course.

Type "?" for the available commands

or "? task" for the current task.

Tutor: We are now at the top level mode.

Your first exercise is to invoke the MM.

%mm

A.ISI.EDU.#Internet MM-20 6.1(1145)

Last read: 5-Jun-88 11:15:20 12 messages, 40 pages

3 messages unseen; 2 deleted

U	10	5-Jul	Tutor	Exercise #10
U	11	7-Jul	Tutor	Delete this file
U	12	10-Jul	Gen. X	Task #12

Tutor: Good. Now, we are at the MM mode.

Lets browse the headers of the current message in the mail box.

MM> ?

Following are the possible commands at mm level:

answer	headers all	list	send
copy	headers current	logout	type
delete	headers delete	move	undelete
exit	headers flag	next	unflag
expunge	headers unseen	previous	
flag	help	quit	
forward	jump	read	

MM> headres current

Ooops! You mean "headers current"? yes

U	12	10-Jul	Gen. X	Task #12
---	----	--------	--------	----------

Tutor: O.K.

Lets browse the content of the current message.

MM> type

Message 12

Mail_From: Gen. X

Date:

Subject: Task #12

To: Kim

This is the content of message number 12.
Since you have seen this message, the unseen flag will be removed.

Tutor: Now, go back to the top level again.

MM> quit

Tutor: All right. We came back to the TOP level.
Now, lets browse all the headers of messages in the mail box.

%headers all

Ooops! You can issue a "headers all" command only in the "MM" level.

Try again.

%mm

A.ISI.EDU.#Internet MM-20 6.1(1145)
Last read: 5-Jun-88 11:15:20 12 messages, 40 pages
2 messages unseen; 2 deleted

U	10	5-Jul	Tutor	Exercise #10
U	11	7-Jul	Tutor	Delete this file

MM> headers all

FLAG	NUMBER	DATE	AUTHOR	SUBJECT
	1	25-Jun	Tutor	Exercise #1
	2	27-Jun	Gen. X	Task #2
D	3	28-Jun	Tutor	Garbage
A	4	29-Jun	Gen. Z	Reply Needed
F	5	30-Jun	Tutor	Special Note
	6	1-Jul	Tutor	Exercise #6
A	7	30-Jun	Gen. Z	Reply Needed
D	8	3-Jul	Somebody	Another Garbage
F	9	5-Jul	Gen. S	Another Special Note
U	10	5-Jul	Tutor	Exercise #10
U	11	7-Jul	Tutor	Delete this file
	12	10-Jul	Gen. X	Task #12

Tutor: Now, list all the headers of flagged messages in the mail box.

MM> jeaders flag

Ooops! You mean "headers flag"? y

FLAG	NUMBER	DATE	AUTHOR	SUBJECT
F	5	30-Jun	Tutor	Special Note
F	9	5-Jul	Gen. S	Another Special Note

Tutor: Good. Now, set the flag on the message number 12.

MM> flag

Message number to be flagged? 12

Tutor: O.K. Now, list all the headers of flagged messages again.

MM> headers flag

FLAG	NUMBER	DATE	AUTHOR	SUBJECT
F	5	30-Jun	Tutor	Special Note
F	9	5-Jul	Gen. S	Another Special Note
F	12	10-Jul	Gen. X	Task #12

Tutor: List all the headers of the unseen messages in the mail box.

MM> headers unseen

FLAG	NUMBER	DATE	AUTHOR	SUBJECT
U	10	5-Jul	Tutor	Exercise #10
U	11	7-Jul	Tutor	Delete this file

Tutor: All right.

Now, lets browse the message number 10.

MM> type 10

Message 10

Mail_From: Tutor

Date:

Subject: Exercise #10

To: Kim

This is the content of message number 10.

Tutor: Now, jump to the message number 7 in the mail box.

MM> jump 7

Tutor: All right.

Now, lets browse the current message in the mail box again.

MM> type

Message 7

Mail_From: Gen. Z

Date:

Subject: Reply Needed

To: Kim

This is the content of message number 7.

Tutor: Good.

Now, browse the next message in the mail box.

MM> next

Message 8

Mail_From: Somebody

Date:

Subject: Another Garbage

To: Kim

This is the content of message number 8.

Tutor: Now, browse the previous message in the mail box.

MM> previuos

Ooops! You mean "previous"? y

Message 7

Mail_From: Gen. Z

Date:

Subject: Reply Needed

To: Kim

This is the content of message number 7.

Tutor: Good. Now, send a message to X@SRI-KL.

Tutor: Now, browse the previous message in the mail box.

MM> previous

Message 11

Mail_From: Tutor

Date:

Subject: Delete this file

To: Kim

This is the content of message number 11.

Tutor: Good. Now, send a message to X@SRI-KL.

MM> send

To: X@SRI-KL

Cc: yee,liang

Subj: Good news

Message (end with ESCAPE or CTRL-D or CTRL-Z): This is my first message.
S>send

Tutor: Excellent.
Now, send a message to the address in message 6

MM> type 6

Message 6
Mail_From: Tutor
Date:
Subject: Exercise #6
To: Kim

Send a message to GK@NPS.ARPA.

MM> send
To: GK@NPS.ARPA
Cc: metin
Subj: Task
Message (end with ESCAPE or CTRL-D or CTRL-Z): I got it, sir.
S>send

APPENDIX B - SOURCE PROGRAMS

```

/*****
/*
/*   Module Name: TOPS20-MM
/*
/*   Author: Capt.Taewoo Kim, ROKA
/*
*****/

/* TOP commands. */

action(top,help) :- respond_help(top).
action(top,?) :- respond_help(top).
action(top,logout) :- halt.

/* topaction('mm') */
action(top,mm) :- change_level(mm),
    date(Date),time(Time),
    countup_messages(message(X,A,B,C,D,E),M),
    countup_unseen(message(u,A,B,C,D,E),U),
    countup_deleted(message(d,A,B,C,D,E),Del),
    nl,write('A.ISI.EDU.#Internet MM-20 6.1(1145)'), nl,
    write(' Last read: '),write(Date),write(Time),write(M),
    write(' messages '),write(' 40 pages '), nl,
    write(U),write(' messages unseen; '),write(Del),
    write(' deleted'),nl,nl,
    headers_unseen.

headers_unseen :-
    message(u,A,B,C,D,E), write_body(A,B,C,D,E), fail.
headers_unseen.

date(' 5-Jun-88 ').
time(' 11:15:20 ').

action(X,trace) :- trace.
action(top,X) :- check_error(top,X).

/* MM level commands. */

/* mmaction('answer') */
action(mm,answer) :- respond_answer.

/* mmaction('copy') */
action(mm,copy) :- respond_copy.

/* mmaction('delete') */
action(mm,delete) :- nl,
    write(' Message number to be deleted? '),niceread(Num),nl,
    convert(Num,Int),
    message(X,A,Int,C,D,E),retract(message(X,A,Int,C,D,E)),
    asserta(message(d,' D ',Int,C,D,E)),nl,sort_messages.

/* mmaction('exit') */
action(mm,exit) :- erase_deleted.

/* mmaction('expunge') */

```

```

action(mm,expunge) :- erase_deleted.

erase_deleted :- nl,retract(message(d,A,B,C,D,E)),fail.
erase_deleted.

/* nmaction('flag') */
action(mm,flag) :- nl,
    write( 'Message number to be flagged? '),niceread(Num),nl,
    convert(Num,Int),
    message(X,A,Int,C,D,E),retract(message(X,A,Int,C,D,E)),
    asserta(message(f,' F',Int,C,D,E)),nl,sort_messages.

/* nmaction('forward') */
action(mm,forward) :- nl,
    write( 'Message number to be forwarded? '),
    niceread(Num),nl,convert(Num,Int),
    message(X,A,Int,C,D,E),
    asserta(fmessage(X,' F',Int,C,D,E)),nl.

/* nmaction('headers all') */
action(mm,'headers all') :-
    write_title, headers_all.
headers_all :-
    message(X,A,B,C,D,E),
    ((X = 0, B < 10, write(' '), write(B), tab(4), write(C), tab(4),
        write(D), tab(4), write(E),tab(4),nl);
    (X = 0, B > 9, write_body(A,B,C,D,E);
    (X=a;X=d;X=f;X=u), write_body(A,B,C,D,E))), fail.
headers_all.

/* nmaction('headers answer') */
action(mm,'headers answer') :-
    write_title, headers_answer.
headers_answer :-
    message(a,A,B,C,D,E), write_body(A,B,C,D,E), fail.
headers_answer.

/* nmaction('headers current') */
action(mm,'headers current') :- current_msg(QMN),
    message(X,A,QMN,C,D,E), write_body(A,QMN,C,D,E).

/* nmaction('headers delete') */
action(mm,'headers delete') :-
    write_title, headers_delete.
headers_delete :-
    message(d,A,B,C,D,E), write_body(A,B,C,D,E), fail.
headers_delete.

/* nmaction('headers flag') */
action(mm,'headers flag') :-
    write_title, headers_flag.
headers_flag :-
    message(f,A,B,C,D,E), write_body(A,B,C,D,E), fail.
headers_flag.

/* nmaction('headers unseen') */
action(mm,'headers unseen') :-
    write_title, headers_unseen.
headers_unseen :-
    message(u,A,B,C,D,E), write_body(A,B,C,D,E), fail.

```

```

headers_unseen.

write_title :- nl,
    write('FLAG NUMBER DATE      AUTHOR      SUBJECT'), nl.

write_body(A,B,C,D,E) :-
    write(A),tab(4), write(B),tab(4),
    write(C),tab(4), write(D),tab(4),write(E),nl.

/* nmaction('help') */
action(nm,help) :- respond_help(nm).
action(nm,?) :- respond_help(nm).

action(Level,Help) :- long_help_command(Help,Command),
    give_def(Level,Command).

/* nmaction('jump') */
action(nm,JC) :- long_jump_command(JC,NM),
    NM > 1, NM < 13, !,
    current_msg(CM), retract(current_msg(CM)),
    asserta(current_msg(NM)).

action(nm,JC) :- long_jump_command(JC,NM),
    write('Message '), write(NM), write(' does not exist.').

/* nmaction('list') */
action(nm,list) :- respond_list.

/* nmaction('logout') */
action(nm,logout) :- nmaction(expunge), halt.

/* nmaction('move') */
action(nm,move) :- respond_move.

/* nmaction('next') */
action(nm,next) :- current_msg(K), K < 12, retract(current_msg(K)),
    Kp1 is K+1, asserta(current_msg(Kp1)), print2(Kp1).
action(nm,next) :- write('Message currently ends at number 12').

/* nmaction('previous') */
action(nm,previous) :- current_msg(K), K > 1, retract(current_msg(K)),
    Km1 is K-1, asserta(current_msg(Km1)), print2(Km1).
action(nm,next) :- write('Message starts from number 1').

/* nmaction('quit') */
action(nm,quit) :- change_level(top).
action(nm,q) :- change_level(top).

/* nmaction('type') */
action(nm,type) :- current_msg(K), print2(K).
action(nm,C) :- current_msg(K3), retract(current_msg(K3)),
    long_type_command(C,K), asserta(current_msg(K)), print2(K).

/* nmaction('undelete') */
action(nm,undelete) :- nl,
    write('Message number to be undeleted? '),
    nicerread(Num),nl,convert(Num,Int),
    message(d,A,Int,B,D,E),retract(message(d,A,Int,B,D,E)),
    asserta(message(0,0,Int,B,D,E)),nl,
    sort_messages.

```

```

/* mmaction('unflag') */
action(mm,unflag) :- nl,
    write( 'Message number to be unflagged? '),
    niceread(Num),nl,convert(Num,Int),
    message(f,A,Int,B,D,E),retract(message(f,A,Int,B,D,E)),
    asserta(message(0,0,Int,B,D,E)),nl,
    sort_messages.

/* Check error */
action(mm,X) :- check_error(mm,X).

/* Commands at SEND level */

/* sendaction('display') */
action(send,display) :- send_display.

/* sendaction('erase') */
action(send,erase) :- send_erase.

/* sendaction('help') */
action(send,help) :- respond_help(send).
action(send,?) :- respond_help(send).

/* sendaction('send') */
action(send,send) :- send_send.

/* sendaction('quit') */
action(send,quit) :- change_level(mm).
action(send,q) :- change_level(mm).

/* sendaction('type') */
action(send,type) :- send_type.

/* Illegal commands */
action(send,X) :- error(X).

/* Print the requested message. */

print2(Num) :- message(X,A,Num,C,D,E),nl,nl,
    write('Message '),write(Num),nl,
    write('Mail_From: '),write(D),nl,
    write('Date: '),nl,
    write('Subject: '),write(E),nl,
    write('To: '),user(Name),nwrite(Name),nl,nl,
    message_content(Num,Content),write(Content),nl,
    check_flag(Flag,A,Num,C,D,E),
    sort_messages, !.

check_flag(u,A,Num,C,D,E) :-
    retract(message(u,A,Num,C,D,E)),
    asserta(message(0,' ',Num,C,D,E)).

check_flag(X,A,Num,C,D,E).

/* count the number of each messages */

countup_messages(message(X,A,B,C,D,E),M) :- asserta(counter(0)),
    call(message(X,A,B,C,D,E)), counter(K), retract(counter(K)),
    K2 is K+1, asserta(counter(K2)), fail.

```

```

countup_messages(message(X,A,B,C,D,E),M) :- counter(M), retract(counter(M)), !.

countup_unseen(message(u,A,B,C,D,E),U) :- asserta(counter(0)),
    call(message(u,A,B,C,D,E)), counter(K), retract(counter(K)),
    K2 is K+1, asserta(counter(K2)), fail.
countup_unseen(message(u,A,B,C,D,E),U) :- counter(U), retract(counter(U)), !.

countup_deleted(message(d,A,B,C,D,E),Del) :- asserta(counter(0)),
    call(message(d,A,B,C,D,E)), counter(K), retract(counter(K)),
    K2 is K+1, asserta(counter(K2)), fail.
countup_deleted(message(d,A,B,C,D,E),Del) :- counter(Del),
    retract(counter(Del)), !.

/* CODES FOR SEND OP. */

action(mm,send) :- change_level(to).
action(to,X) :- asserta(receiver(X)), !.
action(cc,X) :- asserta(ccreceiver(X)), !.
action(subject,X) :- asserta(message_subject(X)), !.
action(message,X) :- asserta(message(X)), !.
action(ready_to_send,send) :- last_sent_msg(LSM),
    current_date(CD), receiver(Rname), message(M),
    assertz(sent_message(s,' ',LSM,Date,Rname,M)),
    retract(last_sent_msg(LSM)), NLSM is LSM + 1,
    assertz(last_sent_msg(NLSM)), retract(receiver(Rname)),
    retract(ccreceiver(CCR)), retract(message(M)),
    retract(message_subject(MS)), !.

current_date(' 14-Apr-88 ').

/* sorts messages with message number */

sort_messages :- bagof(N,A^B^C^D^E^message(A,B,N,C,D,E).L),
    sort(L,SN), sort1(SN), !.

sort1(SN) :- first(SN,F), message(A,B,F,C,D,E), retract(message(A,B,F,C,D,E)),
    assertz(message(A,B,F,C,D,E)), delete(F,SN,NSN), sort1(NSN).
sort1(SN).

first([X|_],X).

delete(X,[],[]).
delete(X,[X|L],M) :- !, delete(X,L,M).
delete(X,[Y|L],[_|M]) :- delete(X,L,M).

/* read in each character and convert it into list of ASCII codes */
/* until carriage return is pressed. */

niceread(L) :- checkretract(readbuff(L2)), asserta(readbuff([])),
    niceread2(L), !.
niceread2(L) :- get0(C), niceread3(C,L).

/* niceread3(32,L) :- niceread2(L). <== ignore the spaces. */

niceread3(10,L) :- !, readbuff(L2), reverse(L2,L).
niceread3(C,L) :- readbuff(L3), retract(readbuff(L3)),
    asserta(readbuff([C|L3])), niceread2(L).

checkretract(S) :- call(S), retract(S), !.

```

```

checkretract(S).

reverse(L,R) :- reverse2(L,[],R).
reverse2([],L,L) :- !.
reverse2([X|L],R,S) :- reverse2(L,[X|R],S).

convert([C|T],Int) :- T = [], Int is C-48.
convert([C|T],Int) :- T > 47, T < 58, Int is (((C-48)*10)+(T-48)).
convert([C|T],Int) :- nwrite([C|T]),write(' <== Invalid input!'),nl.

/* code for reading the message user types in as a list of lists. */

read_message(L) :-
    write('Please type your message; terminate with a control-X.'), nl,
    write('                                and a carriage return.'), nl,
    read_message2(L).
read_message2([X|L]) :- niceread(X), read_message3(X,L).
read_message3(X,[]) :- termination(X), !.
read_message3(X,L) :- read_message2(L).
termination(L) :- last(L,24).
last([X],X) :- !.
last([X|L],Y) :- last(L,Y).

/* write the message list ==> each sublist as a separate line. */
write_message([]) :- !.
write_message([X|L]) :- writel(X), nl, write_message(L).

writel([]).
writel([X|L]) :- put(X), writel(L).

/* write the list of ASCII code into character string. */
nwrite([]).
nwrite([X|R]) :- put(X), nwrite(R).

/* message headers */
message(0, '1, '25-Jun', 'Tutor', 'Exercise #1').
message(0, '2, '27-Jun', 'Gen. X', 'Task #2').
message(d, 'D', '3, '28-Jun', 'Tutor', 'Garbage').
message(a, 'A', '4, '29-Jun', 'Gen. Z', 'Reply Needed').
message(f, 'F', '5, '30-Jun', 'Tutor', 'Special Note').
message(0, '6, '1-Jul', 'Tutor', 'Exercise #6').
message(a, 'A', '7, '30-Jun', 'Gen. Z', 'Reply Needed').
message(d, 'D', '8, '3-Jul', 'Somebody', 'Another Garbage').
message(f, 'F', '9, '5-Jul', 'Gen. S', 'Another Special Note').
message(u, 'U', '10, '5-Jul', 'Tutor', 'Exercise #10').
message(u, 'U', '11, '7-Jul', 'Tutor', 'Delete this file').
message(u, 'U', '12, '10-Jul', 'Gen. X', 'Task #12').

/* contents of messages */
message_content(1, 'This is the content of message number 1.').
message_content(2, 'This is the content of message number 2.').
message_content(3, 'This is the content of message number 3.').
message_content(4, 'This is the content of message number 4.').
message_content(5, 'This is the content of message number 5.').
message_content(6, 'Send a message to GENPS.ARPA.').
message_content(7, 'This is the content of message number 7.').
message_content(8, 'This is the content of message number 8.').
message_content(9, 'This is the content of message number 9.').
message_content(10, 'This is the content of message number 10.').
message_content(11, 'This is the content of message number 11.').

```


message_content(12, ' This is the content of message number 12.
Since you have seen this message, the unseen flag will be removed.').

space1(' ').
space2(' ').
space3(' ').
space4(' ').

```

/*****
/*
/*   Module Name: TUTOR
/*
/*   Author: Prof. Neil C. Rowe
/*
/*
/*****

/* Problem-independent code for "means-ends tutoring": tutoring for */
/* learning of sequences modelable by means-ends analysis. */

/* For an application, you must define: */
/* (1) recommended(<difference>,<operator>) --recommendation conditions */
/* (2) precondition(<operator>,<factlist>) --precondition facts */
/* (3) deletepostcondition(<operator>,<factlist>) or */
/*     deletepostcondition(<operator>,<conditionlist>,<factlist>) */
/* --gives facts deleted by op.; 3-arg. form requires additional facts true */
/* (4) addpostcondition(<operator>,<factlist>) or */
/*     addpostcondition(<operator>,<conditionlist>,<factlist>) */
/* --gives facts added by op.; 3-arg. form requires additional facts true */

/* Some optional definitions you may include: */
/* (5) randsubst(<op>,<substlist1>,<substlist2>,...) */
/* --gives random-substitution triples or quadruples, each in the form: */
/* [initial-fact,<ending-fact>,<transition-prob>,<message to user> */
/*     Note: first and second arguments can be the word "none"; */
/*     fourth argument is optional. */
/* (6) nopref(<operator1>,<operator2>) --if the order (priority) of two */
/*     operators in the "recommended" rules was arbitrary, include this fact */
/* (7) intro(<text>) --introductory info for student */
/* (8) debugflag --if asserted, debugging info printed re means-ends anal. */

tutor(State,Goal) :-
    not(check_obvious_errors), issue_warnings, randinit,
    uniqueassert(top_goal(Goal)),
    bagof(X,P^precondition(X,P),XL), uniqueassert(op_list(XL)),
    once_means_ends(State,Goal,Oplist2,Goalstate2),
    uniqueassert(top_solution(State)), abolish(mainline_states,4),
    means_ends_tutor(State,Goal,Oplist,Goalstate,[]),
    nl, nl, !.

tutor(State,Goal) :-
    write('Too bad: a solution is now impossible.'), nl, !.

means_ends_tutor(State,Goal,[],State,Stack) :- difference(Goal,State,[]), !.

means_ends_tutor(State,Goal,Oplist,State,Stack) :- member([State,Goal],Stack),
    !, fail.

means_ends_tutor(State,Goal,Oplist,Goalstate,Stack) :-
    not(once_means_ends(State,Goal,Oplist,Goalstate)), !, fail.

means_ends_tutor(State,Goal,Oplist,Goalstate,Stack) :-
    difference(Goal,State,D),
    applicable_op(D,Op),
    precondition(Op,Prelist),
    all_achievable(State,Prelist), !,
    means_ends_tutor(State,Prelist,Preoplist,Prestate,[[State,Goal]|Stack]),
    !, met(State,Goal,Oplist,Goalstate,Stack,Prelist,Preoplist,Prestate,Op,D).

```

```

met(State,Goal,Preoplist,Prestate,Stack,Prelist,Preoplist,Prestate,Op,D) :-
    difference(Goal,Prestate,[]), !.

met(State,Goal,Oplist,Goalstate,Stack,Prelist,Preoplist,Prestate,Op,D) :-
    difference(Goal,Prestate,D2), not(applicable_op(D2,Op)), !,
    means_ends_tutor(Prestate,Goal,Oplist2,Goalstate,[]),
    append(Preoplist,Oplist2,Oplist).

met(State,Goal,Oplist,Goalstate,Stack,Prelist,Preoplist,Prestate,Op,D) :-
    check_with_student(Op,Prestate,D,NewOp),
    get_deletepostcondition(NewOp,Prestate,Deletepostlist),
    deleteitems(Deletepostlist,Prestate,Prestate2),
    get_addpostcondition(NewOp,Prestate,Addpostlist),
    union(Addpostlist,Prestate2,Postlist2),
    do_randsubst(NewOp,Postlist2,Postlist),
    check_mainline_return(Postlist), !,
/* Note last arg. below empty to allow for randsubst returning to past state */
    means_ends_tutor(Postlist,Goal,Postoplist,Goalstate,[]),
    append(Preoplist,[NewOp|Postoplist],Oplist).

do_intro :- intro(T), write(T), nl, !.
do_intro.

/* Problem-definition errors */

check_obvious_errors :- setof([M,A],obvious_error(M,A,MAL), !,
    writepairlist(MAL).

obvious_error('precondition fact missing for operator ',O) :-
    recommended(D,O), not(precondition(O,L)).

obvious_error('deletepostcondition fact missing for operator ',O) :-
    recommended(D,O), not(get_deletepostcondition(O,S,L)).

obvious_error('addpostcondition fact missing for operator ',O) :-
    recommended(D,O), not(get_addpostcondition(O,S,L)).

obvious_error('"recommended" fact missing for operator ',O) :-
    precondition(C,!), not(recommended(D,O)).

obvious_error('"recommended" fact missing for operator ',O) :-
    get_deletepostcondition(O,S,L), not(recommended(D,O)).

obvious_error('"recommended" fact missing for operator ',O) :-
    get_addpostcondition(O,S,L), not(recommended(D,O)).

issue_warnings :- setof([M,A],possible_error(M,A,MAL), !,
    write('Warnings:'), nl, writepairlist(MAL), nl.
issue_warnings.

possible_error('This fact is not creatable: ',F) :- precondition(O,PL),
    backtracking_member(F,PL), uncreatable(F).

writepairlist([]).
writepairlist([[X,Y]|L]) :- write(X), write(Y), nl, writepairlist(L).

/* Handling of randomness */

do_randsubst(O,S,NS) :- randsubst(O,RL), !, do_randsubst2(RL,S,NS).
do_randsubst(O,S,S).

```

```

do_randsubst2([],S,S).
do_randsubst2([F,NF,P]|L,S,NS) :- random(1000,K), P1000 is P*1000,
    K=<P1000, changestate(F,NF,S,S2), !, do_randsubst2(L,S2,NS).
do_randsubst2([F,NF,P,M]|L,S,NS) :- random(1000,K), P1000 is P*1000,
    K=<P1000, changestate(F,NF,S,S2), !, write(M), nl, do_randsubst2(L,S2,NS).
do_randsubst2([C|L],S,NS) :- do_randsubst2(L,S,NS).

changestate(none,NF,S,[NF|S]) :- !, not(member(NF,S)),
    write('Random change made: '), nl, !.
changestate(F,none,S,S2) :- !, member(F,S),
    write('Random change made: '), nl, !.
changestate(F,NF,S,[NF|S3]) :- !, member(F,S),
    write('Random change made: '), nl, !.

permutation([],[]) :- !.
permutation(L,[I|PL]) :- randitem(L,I), delete(I,L,L2), permutation(L2,PL).

randitem(L,I) :- length(L,N), random(N,K), item(K,L,I).

randinit :- C is cputime*1000, PC is floor(C), S is PC mod 2311,
    uniqueassert(randseed(S)).

random(N,K) :- randseed(S), nexttrand(S,NS), K is NS mod N,
    retract(randseed(S)), asserta(randseed(NS)).

nexttrand(S,NS) :- FIC is 100*cputime, IC is floor(FIC),
    S2 is ((S*S)+IC) mod 2311, S3 is (S2*S2) mod 2311, NS is (S3*S) mod 2311.

item(K,[],_) :- !, fail.
item(K,[X|L],X) :- K=<1, !.
item(K,[X|L],Y) :- Kml is K-1, item(Kml,L,Y).

/* Tutoring rules */
/* handle_student_op rules have been added to the original code */

check_with_student(O,S,D,NO) :-
    member(level(Current_Level),S),
    temprompt(Current_Level,P), write(P),
    niceread(LCommand), name(SCommand,LCommand),
    O2 =.. [command,Current_Level,SCommand],
    handle_student_op(O2,O,S,D,NO), !.

handle_student_op(O,O,S,D,O) :- O =.. [P,CL,O3], !, action(CL,O3), !.

handle_student_op(O2,O,S,D,NO) :- O2 =.. [P,CL,C], helpword(C), !,
    respond_help(CL), !, check_with_student(O,S,D,NO).

handle_student_op(O2,O,S,D,NO) :- O2 =.. [P,CL,C], help_task(C), !,
    check_with_student(O,S,D,NO).

handle_student_op(O2,O,S,D,O) :- O2 =.. [P,L,C], mstop(C), !, halt.

handle_student_op(O2,O,S,D,O) :- O2 =.. [P,L,C], mabort(C), !, abort.

handle_student_op(O2,O,S,D,NO) :- O2 =.. [P,L,C], mtrace(C), !, trace,
    check_with_student(O,S,D,NO).

handle_student_op(O2,O,S,D,NO) :- level(Level), check_error(Level,O2,NO), !,
    check_with_student(O,S,D,NO).

```

```

handle_student_op(O2,O,S,D,NO) :- op_list(OL), not(singlemember(O2,OL)), !,
    write('Tutor: Not a valid operator--please choose one of the following: '),
    level(CL),
    respond_help(CL), check_with_student(O,S,D,NO).

handle_student_op(O2,O,S,D,NO) :- precondition(O2,PO2), difference(PO2,S,D2),
    not(D2=[]), !, write('Tutor: That operator requires that '),
    writelist(D2,precond), write(' '), nl,
    check_with_student(O,S,D,NO).

handle_student_op(O2,O,S,D,NO) :- apply_op(O2,S,S),
    write('Tutor: That will not affect anything. '),nl,
    check_with_student(O,S,D,NO).

handle_student_op(O2,O,S,D,NO) :- apply_op(O2,S,S2), top_goal(G),
    not(once_means_ends(S2,G,OL2,GS2)), !,
    write('Tutor: You cannot ever succeed if you do that. '), nl,
    check_with_student(O,S,D,NO).

handle_student_op(O2,O,S,D,O2) :- top_goal(G), apply_op(O,S,S3),
    apply_op(O2,S,S2), compare_solutions(S3,G,OL3,GS3,S2,G,OL2,GS2),
    subsequence([O|OL3],OL2), !, apply_ops([O|OL3],S,SL,GS4),
    elimdups(SL,ESL), asserta(mainline_states(ESL,O2,S,O)),
    write('Tutor: That does not seem immediately helpful, but I will try it. '),
    nl, O2 =.. [P,CL,O3], !, action(CL,O3).

handle_student_op(O2,O,S,D,O2) :- (nopref(O2,O):nopref(O,O2)), !,
    write('O.K. '), nl, O2=..[P,CL,O3], action(CL,O3).

handle_student_op(O2,O,S,D,O2) :- top_goal(G),
    once_means_ends(S,G,OL,FS), not(member(O2,OL)), !,
    write('Tutor: I will try it, but it is not recommended for the problem. '),
    nl, O2 =.. [P,CL,O3], !, action(CL,O3).

handle_student_op(O2,O,S,D,O2) :- top_goal(G), difference(G,S,D2),
    all_achievable(S,D2), applicable_op(D2,O3), precondition(O3,PL),
    least_common_op(S,G,O,O2,PL,Groot), !,
    write('Tutor: I will try it, but it is not recommended first when '),
    difference(Groot,S,D5), delete_uncreatable(D5,D6),
    permutation(D6,D7), writelist(D7,precond), write(' '), nl,
    O2 =.. [P,CL,O3], !,
    action(CL,O3).

handle_student_op(O2,O,S,D,O2) :-
    write('Tutor: Not the operator I would choose, but let us try it. '), nl,
    O2 =.. [P,CL,O3], !,
    action(CL,O3).

temprompt(top,'%').
temprompt(mm,'MM> ').
temprompt(to,'To: ').
temprompt(cc,'Cc: ').
temprompt(subject,'Subj: ').
temprompt(message,'Message (end with ESCAPE or CTRL-D or CTRL-Z): ').
temprompt(ready_to_send,'S>').

helpword(help).
helpword(h).
helpword(huh).

```

```

helpword(?).
mstop(halt).
mstop(logout).
mabort(abort).
help_task('?task').
mtrace(trace).

/* Intermediate predicates used by the tutor */

least_common_op(S,G,O,O2,G2,G) :- once_means_ends(S,G2,OL,NS),
    (not(member(O,OL)); not(member(O2,OL))), !.

least_common_op(S,G,O,O2,G2,Droot) :- difference(G2,S,D), all_achievable(S,D),
    applicable_op(D,O3), precondition(O3,G3),
    least_common_op(S,G2,O,O2,G3,Droot), !.

compare_solutions(S3,G,OL3,GS3,S2,G,OL2,GS2) :-
    once_means_ends(S3,G,OL3,GS3),
    once_means_ends(S2,G,OL2,GS2), !.

cache_states(S,G,[],GS) :- !.
cache_states(S,G,OL,GS) :- cached(S,G,OL,GS), !.
cache_states(S,G,OL,GS) :- cached(S2,G2,OL2,GS2), check_permutation(S,S2),
    check_permutation(G,G2), !.
cache_states(S,G,[O|OL],GS) :- asserta(cached(S,G,[O|OL],GS)),
    apply_op(O,S,NS), cache_states(NS,G,OL,GS), !.

apply_ops([],S,[S],S) :- !.
apply_ops([O|OL],S,[S|SL],NS) :- apply_op(O,S,S2), apply_ops(OL,S2,SL,NS).
apply_op(O,S,NS) :- get_deletepostcondition(O,S,DP), deleteitems(DP,S,S2),
    get_addpostcondition(O,S,AP), union(AP,S2,NS), !.

check_mainline_return(S) :- mainline_states(SL,O,OS,BO),
    check_mainline_return2(S,SL,O,OS,BO).
check_mainline_return(S).

check_mainline_return2(S,[S2|SL],O,OS,BO) :- permute_member(S,[S2]),
    !, write('You are returning to a previous state. '), nl.

check_mainline_return2(S,SL,O,OS,BO) :- permute_member(S,SL), !,
    write('Do you see now that your choice of the '), write(O),
    write(' action in the state with the facts '), writelist(OS,state),
    write(' was not the best choice; the '), write(BO),
    write(' action would have been better. '), nl,
    retract(mainline_states(SL,O,OS,BO)).

/* Natural language output */

writelist([],R) :- !.
writelist([X],R) :- !, writefact(X,R).
writelist([X,Y],R) :- !, writefact(X,R), write(' and '), writefact(Y,R).
writelist(L,R) :- writelist2(L,R).
writelist2([X],R) :- !, write('and '), writefact(X,R).
writelist2([X|L],R) :- writefact(X,R), write(', '), writelist2(L,R).
writefact(F,state) :- atom(F), write(F), write(' is true'), !.
writefact(not(F),state) :- atom(F), !, write(F), write(' is false'), !.
writefact(not(F),state) :- F=..[P,X], atom(X), !, write(X), is_form(X,IX),
    write(IX), write('not '), write(P), !.
writefact(not(F),state) :- F=..[P,X], !, writefact(X), is_form(X,IX).

```

```

write(IX), write('not '), write(P), !.
writefact(not(F),state) :- F=..[P,X,Y], !, write(X), write(' not '),
write(P), write(' '), write(Y), !.
writefact(F,state) :- F=..[P,X], atom(X), !, write(X), is_form(X,IX),
write(IX), write(P), !.
writefact(F,state) :- F=..[P,X], !, writefact(X,state), is_form(X,IX),
write(IX), write(P), !.
writefact(F,state) :- F=..[P,X,Y], !, write(X), write(' '),
write(P), write(' '), write(Y), !.
writefact(F,precond) :- atom(F), write(F), write(' must be true'), !.
writefact(not(F),precond) :- atom(F), !, write(P), write(' must be false'), !.
writefact(not(F),precond) :- F=..[P,X], atom(X), !, write(X),
write(' must not be '), write(P), !.
writefact(not(F),precond) :- F=..[P,X], !, writefact(X,state),
write(' must not be '), write(P), !.
writefact(not(F),precond) :- F=..[P,X,Y], !, write(X), write(' must not be '),
write(P), write(' '), write(Y), !.
writefact(F,precond) :- F=..[P,X], atom(X), !, write(X),
write(' must be '), write(P), !.
writefact(F,precond) :- F=..[P,X], !, writefact(X,state),
write(' must be '), write(P), !.
writefact(F,precond) :- F=..[P,X,Y], write(X), write(' must be '), write(P),
write(' '), write(Y), !.
writefact(F,op) :- write(F), !.
writefact(F,R) :- write(F).
is_form(X,' is ') :- not(atom(X)), !.
is_form(X,' are ') :- name(X,NX), last(NX,115), !.
is_form(X,' is ').

/* The original means-ends program (used for "what if" reasoning) */

once_means_ends(State,Goal,Oplist,Goalstate) :-
means_ends(State,Goal,Oplist,Goalstate),
cache_states(State,Goal,Oplist,Goalstate), !.

means_ends(State,Goal,Oplist,Goalstate) :-
means_ends2(State,Goal,Oplist,Goalstate,[]), writedebug7.

means_ends2(State,Goal,Oplist,Goalstate,Stack) :-
cached(State2,Goal2,Oplist,Goalstate), check_permutation(Goal,Goal2),
check_permutation(State,State2), !, writedebug6(Stack), !.

means_ends2(State,Goal,Oplist,Goalstate,Stack) :- member([State,Goal],Stack),
!, writedebug4(Stack), fail.

means_ends2(State,Goal,[],State,Stack) :- difference(Goal,State,[]), !.

means_ends2(State,Goal,Oplist,Goalstate,Stack) :- difference(Goal,State,D),
applicable_op(D,Operator), precondition(Operator,Prelist),
all_achievable(State,Prelist), writedebug1(D,Operator,Stack),
means_ends2(State,Prelist,Preoplist,Prestate,[[State,Goal]|Stack]),
writedebug2(Prestate,D,Operator,Stack),
get_deletepostcondition(Operator,Prestate,Deletepostlist),
deleteitems(Deletepostlist,Prestate,Prestate2),
get_addpostcondition(Operator,Prestate,Addpostlist),
union(Addpostlist,Prestate2,Postlist),
means_ends2(Postlist,Goal,Postoplist,Goalstate,[[State,Goal]|Stack]),
writedebug3(Goalstate,Operator,Stack),
append(Preoplist,[Operator|Postoplist],Oplist).

```

```

means_ends2(State,Goal,Oplist,Goalstate,Stack) :-
    writedebug5(State,Goal,Stack), !, fail.

/* Debugging tools */

writedebug1(D,O,Stack) :- not(debugflag), !.
writedebug1(D,O,Stack) :- length(Stack,Nml), N is Nml+1, write('>>Operator '),
    write(O), write(' suggested at level '), write(N), nl,
    write('to achieve difference of {'), writelist(D,state), write('}'), nl, !.

writedebug2(S,D,O,Stack) :- not(debugflag), !.
writedebug2(S,D,O,Stack) :- length(Stack,Nml), N is Nml+1,
    write('>>Operator '), write(O), write(' applied at level '), write(N), nl,
    write('to reduce difference of {'), writelist(D,state), write('}'), nl,
    write('in state in which '), writelist(S,state), nl, !.

writedebug3(S,O,Stack) :- not(debugflag), !.
writedebug3(S,O,Stack) :- length(Stack,Nml), N is Nml+1, write('>>Level '),
    write(N), write(' terminated at state in which '), writelist(S,state), nl, !.

writedebug4(Stack) :- not(debugflag), !.
writedebug4(Stack) :-
    write('>>>>Reasoning found a potential infinite loop at level '),
    length(Stack,Nml), N is Nml+1, write(N), nl, !.

writedebug5(State,Goal,Stack) :- not(debugflag), !.
writedebug5(State,Goal,Stack) :- write('>>>>Unsolvable problem at level '),
    length(Stack,Nml), N is Nml+1, write(N), nl, write('for state '),
    writelist(State,state), nl, write('and goal '), writelist(Goal,state), nl, !.

writedebug6(Stack) :- not(debugflag), !.
writedebug6(Stack) :- write('>>>>Previously computed solution used at level '),
    length(Stack,Nml), N is Nml+1, write(N), nl, !.

writedebug7 :- not(debugflag), !.
writedebug7 :- nl, !.

/* Miscellaneous utility functions */

delete_uncreatable([],[]).
delete_uncreatable([X|L],M) :- uncreatable(X), !, delete_uncreatable(L,M).
delete_uncreatable([X|L],[X|M]) :- delete_uncreatable(L,M).

all_achievable(S,G) :- difference(G,S,D), not(unachievable_member(D)).

unachievable_member(D) :- backtracking_member(F,D), uncreatable(F).

uncreatable(F) :- precondition(O,L), backtracking_member(F,L),
    not(in_postcondition(F)).

in_postcondition(not(F)) :- any_deletepostcondition(O,DPL), member(F,DPL).
in_postcondition(not(F)) :- randsubst(O,RSL), member([F,X,Y,Z],RSL).
in_postcondition(F) :- not(F=..[not,P]), any_addpostcondition(O,APL),
    member(F,APL).
in_postcondition(F) :- not(F=..[not,P]), randsubst(O,RSL),
    member([X,F,Y,Z],RSL).

any_deletepostcondition(O,L) :- deletepostcondition(O,C,L).
any_deletepostcondition(O,L) :- deletepostcondition(O,L).

```



```

any_addpostcondition(O,L) :- addpostcondition(O,C,L).
any_addpostcondition(O,L) :- addpostcondition(O,L).

get_deletepostcondition(O,S,L) :- deletepostcondition(O,C,L),
    factssubset(C,S), !.
get_deletepostcondition(O,S,L) :- deletepostcondition(O,L).

get_addpostcondition(O,S,L) :- addpostcondition(O,C,L), factssubset(C,S), !.
get_addpostcondition(O,S,L) :- addpostcondition(O,L).

applicable_op(D,O) :- subset(D2,D), recommended(D2,O).

difference([],S,[]).
difference([not(P)|G],S,G2) :- not(singlemember(P,S)), !, difference(G,S,G2).
difference([P|G],S,G2) :- singlemember(P,S), !, difference(G,S,G2).
difference([P|G],S,[P|G2]) :- difference(G,S,G2).

subset([],L).
subset([X|L],L2) :- singlemember(X,L2), subset(L,L2).

factssubset([],L).
factssubset([not(P)|L],L2) :- not(singlemember(P,L2)), !, factssubset(L,L2).
factssubset([not(P)|L],L2) :- !, fail.
factssubset([P|L],L2) :- singlemember(P,L2), factssubset(L,L2).

member(X,L) :- singlemember(X,L).

singlemember(X,[X|L]) :- !.
singlemember(X,[Y|L]) :- singlemember(X,L).

append([],L,L).
append([X|L],L2,[X|L3]) :- append(L,L2,L3).

union([],L,L).
union([X|L1],L2,L3) :- singlemember(X,L2), !, union(L1,L2,L3).
union([X|L1],L2,[X|L3]) :- union(L1,L2,L3).

deleteitems([],L,L).
deleteitems([X|L],L2,L3) :- delete(X,L2,L4), deleteitems(L,L4,L3).
delete(X,[],[]).
delete(X,[X|L],M) :- !, delete(X,L,M).
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

check_permutation(L,M) :- subset(L,M), subset(M,L), !.

subsequence([],L) :- !.
subsequence([X|L],[X|M]) :- !, subsequence(L,M).
subsequence(L,[X|M]) :- subsequence(L,M).

permutelement(X,[X|L]) :- !.
permutelement(X,[Y|L]) :- subset(X,Y), subset(Y,X), !.
permutelement(X,[Y|L]) :- permutelement(X,L).

last([X],X).
last([X|L],Y) :- last(L,Y).

elimdups([],[]).
elimdups([X|L],M) :- singlemember(X,L), !, elimdups(L,M).
elimdups([X|L],[X|M]) :- elimdups(L,M).

```

```
uniqueassert(Q) :- Q=..[P|L], length(L,N), abolish(P,N), asserta(Q).
```

```
backtracking_member(X,[X|L]).
```

```
backtracking_member(X,[Y|L]) :- backtracking_member(X,L).
```

```

/*****
/*                                     */
/*  Module Name: TASKS                */
/*                                     */
/*  Author: Capt.Taewoo Kim, ROKA     */
/*                                     */
*****/

set_up :- reconsult(tops20-mm),
          reconsult(tasks),
          reconsult(tutor),
          save(savedstate),
          go.

go :- initialize,
      greeting,
      introduction,
      issue_task,
      go.

initialize :- asserta(level(top)),
              asserta(last_sent_msg(77)),
              asserta(current_msg(12)),
              asserta(current_task(1)).

greeting :- nl,write('Tutor: Hi, what is your name? '),nicerread(Name),
            asserta(user(Name)), nl,
            write('Tutor: Nice to meet you, '),nwrite(Name),write('.').nl.

introduction :- introduction(I), nl, write(I), nl.
introduction(' Welcome to TOPS20 MM Self-learning course.

Type "?" or "help" for the available commands
and "? task" for the current task.').

issue_task :- bagof(X,P^precondition(X,P),XL), uniqueassert(op_list(XL)),
              task(1).

task(N) :- nl,nl,
           retract(current_task(CT)), asserta(current_task(N)),
           q(N,Q), write(Q), nl,nl,
           task1(N),
           NN is N + 1, task(NN).

change_level(NL) :- retract(level(OL)), asserta(level(NL)).

top_prompt :- nl, write('@ ').
mm_prompt :- nl, write('MM> ').
send_prompt :- nl, write('S> ').
read_prompt :- nl, write('R> ').

/* Basic tasks. */

q(1,'Tutor: We are now at the top level mode.
Your first exercise is to invoke the MM.').
task1(1) :- tutor([level(top)],[level(mm)]).

q(2,'Tutor: Good. Now, we are at the MM mode.
Lets browse the headers of the current message in the mail box.').

```

```

task1(2) :- tutor([level(nm)], [issued_command(nm, 'headers current')]).

q(3, 'Tutor: O.K.
      Lets browse the content of the current message.').
task1(3) :- current_msg(K), tutor([level(nm)], [seen(K)]).

q(4, 'Tutor: Now, go back to the top level again.').
task1(4) :- tutor([level(nm)], [level(top)]).

q(5, 'Tutor: All right. We came back to the TOP level.
      Now, lets browse all the headers of messages in the mail box.').
task1(5) :- tutor([level(top)], [level(nm)]).

q(6, '').
task1(6) :- tutor([level(nm)], [issued_command(nm, 'headers all')]).

q(7, 'Tutor: Now, list all the headers of flagged messages in the mail box.').
task1(7) :- tutor([level(nm)], [issued_command(nm, 'headers flag')]).

q(8, 'Tutor: Good. Now, set the flag on the message number 12.').
task1(8) :- tutor([level(nm)], [issued_command(nm, flag)]).

q(9, 'Tutor: O.K. Now, list all the headers of flagged messages again.').
task1(9) :- tutor([level(nm)], [issued_command(nm, 'headers flag')]).

q(10, 'Tutor: List all the headers of the unseen messages in the mail box.').
task1(10) :- tutor([level(nm)], [issued_command(nm, 'headers unseen')]).

q(11, 'Tutor: All right.
      Now, lets browse the message number 10.').
task1(11) :- tutor([level(nm)], [seen(10)]).

q(12, 'Tutor: Now, jump to the message number 7 in the mail box.').
task1(12) :- tutor([level(nm)], [moved(7)]).

q(13, 'Tutor: All right.
      Now, lets browse the current message in the mail box again.').
task1(13) :- current_msg(K), tutor([level(nm)], [seen(K)]).

q(14, 'Tutor: Good.
      Now, browse the next message in the mail box.').
task1(14) :- tutor([level(nm)], [issued_command(nm, next)]).

q(15, 'Tutor: Now, browse the previous message in the mail box.').
task1(15) :- tutor([level(nm)], [issued_command(nm, previous)]).

q(16, 'Tutor: Good. Now, send a message to X@SRI-KL.').
task1(16) :- tutor([level(nm)], [issued_command(to, 'X@SRI-KL'),
      issued_command(ready_to_send, send)]).

q(17, 'Tutor: Excellent.
      Now, send a message to the address in message 6').
task1(17) :- tutor([level(nm)],
      [seen(6), issued_command(to, A), issued_command(ready_to_send, send)]).

/* Help message */

respond_help(Level) :- nl,

```

```

write(' Following are the possible commands at '),
write(Level), write(' level: '), nl,nl,
level_commands(Level).

```

```

level_commands(top) :-
    write('          mm '), nl,
    write('          quit.'), nl,
    nl.

```

```

level_commands(mm) :-
    write('      answer      headers all      list      send      '), nl,
    write('      copy        headers current  logout    type      '), nl,
    write('      delete       headers delete   move      undelete  '), nl,
    write('      exit         headers flag     next      unflag    '), nl,
    write('      expunge      headers unseen   previous  '), nl,
    write('      flag         help              quit      '), nl,
    write('      forward     jump               read      '), nl,
    nl.

```

```

level_commands(read) :-
    write('      copy      help      quit      unflag '), nl,
    write('      delete    list      reply     '), nl,
    write('      flag      move      send      '), nl,
    write('      forward   next      undelete  '), nl,
    nl.

```

```

level_commands(send) :-
    write('      display      headers      send '), nl,
    write('      erase        quit        type  '), nl,
    nl.

```

```

/* Functions of each command */

```

```

give_def(mm,quit) :- write(' This is the function of command "quit".').

```

```

/* Operator definitions for means-ends analysis */

```

```

/* Recommended operators for required goals. */

```

```

recommended([issued_command(X,Y)],command(X,Y)).

```

```

recommended([level(top)],command(mm,X)) :- member(X,[quit,q]).

```

```

recommended([level(mm)],command(top,mm)).

```

```

recommended([level(to)],command(mm,send)).

```

```

recommended([level(cc)],command(to,X)).

```

```

recommended([level(subject)],command(cc,X)).

```

```

recommended([level(message)],command(subject,Y)).

```

```

recommended([level(ready_to_send)],command(message,X)).

```

```

recommended([moved(K)],command(mm,C)) :- not(var(K)), name('jump ',A1),
name(K,A2), append(A1,A2,AC), name(C,AC).

```

```

recommended([seen(K)],command(mm,type)) :- not(var(K)), current_msg(K).

```

```

recommended([seen(K)],command(mm,C)) :- not(var(K)), name('type ',A1),
name(K,A2), append(A1,A2,AC), name(C,AC).

```

```

/* Preconditions for each operator. */

```

```

precondition(command(L,X),[level(L)]).

/* Deletepostconditions and Addpostconditions for each operators. */

deletepostcondition(command(nm,Y),[]) :- ordcommands(O), member(Y,O).
addpostcondition(command(nm,Y),[issued_command(X,Y)]) :-
    ordcommands(O), member(Y,O).

deletepostcondition(command(nm,Y),[]) :- not(var(Y)), long_jump_command(Y,Z).
addpostcondition(command(nm,C),[moved(K),issued_command(nm,C)])
    :- long_jump_command(C,K).

deletepostcondition(command(nm,Y),[]) :- not(var(Y)), long_type_command(Y,Z).
addpostcondition(command(nm,C),[seen(K),issued_command(nm,C)])
    :- long_type_command(C,K).

deletepostcondition(command(nm,type),[]).
addpostcondition(command(nm,type),[seen(K),issued_command(nm,type)])
    :- current_msg(K).
deletepostcondition(command(top,nm),[level(top)]).
addpostcondition(command(top,nm),[issued_command(top,nm),level(nm)]).

deletepostcondition(command(nm,quit),[level(nm)]).
addpostcondition(command(nm,quit),[issued_command(nm,quit),level(top)]).

deletepostcondition(command(nm,q),[level(nm)]).
addpostcondition(command(nm,q),[issued_command(nm,quit),level(top)]).

deletepostcondition(command(nm,send),[level(nm)]).
addpostcondition(command(nm,send),[issued_command(nm,send),level(top)]).

deletepostcondition(command(X,Y),[level(X)]) :-
    member(X,[to,cc,subject,message,ready_to_send]).
addpostcondition(command(to,X),[level(cc),issued_command(to,X)]).
addpostcondition(command(cc,X),[level(subject),issued_command(cc,X)]).
addpostcondition(command(subject,X),[level(message),issued_command(subject,X)]).
addpostcondition(command(message,X),
    [level(ready_to_send),issued_command(message,X)]).
addpostcondition(command(ready_to_send,send),
    [level(nm),issued_command(ready_to_send,send)]).
addpostcondition(command(ready_to_send,quit),
    [issued_command(ready_to_send,quit),level(ready_to_send)]).

ordcommands([answer,copy,delete,exit,expunge,flag,forward,'headers all',
    'headers current','headers delete','headers flag','headers unseen',
    help,list,logout,move,next,previous,undelete,unflag]).

long_jump_command(C,K) :- not(var(C)), name(C,AC), name('jump ',AT),
    append(AT,AK,AC), name(K,AK), number(K).

long_type_command(C,K) :- not(var(C)), name(C,AC), name('type ',AT),
    append(AT,AK,AC), name(K,AK), number(K).

long_help_command(C,HC) :- not(var(C)), name(C,AC), name('help ',AT),
    append(AT,AK,AC), name(HC,AK).

nopref(O,command(nm,type)) :- O=..[command,nm,C], long_type_command(C,K),
    current_msg(K).
nopref(O,command(nm,jump)) :- O=..[command,nm,C], long_jump_command(O,K),
    current_msg(K).

```

`nopref(q.quit).`
`nopref(n.next).`
`nopref(p.previous).`

```

/*****
/*
/*  Module Name: CORRECT_ERROR
/*
/*  Author: Capt. Taewoo Kim, ROKA
/*
/*
*****/

/* Check against the possible error lists. */

check_error(Level,Ncommand,NO) :- Ncommand =.. [Pred,L,Command],
                                not(valid(Level,Command)),
                                check_error1(Level,Command,NNO),
                                call(Level,NNO,NO).

call(Level,NNO,NO) :- NO =.. [command,Level,NNO].

/* checks error for short type errors */
check_error1(Level,Scommand,Ncommand) :- type_error_list(Level,Ncommand,List),
                                         member(Scommand,List),
                                         ask_error(Level,Ncommand).

type_error_list(Level,q,[a,s,w,l,2]).
type_error_list(top,mm,[nn,jj,mj,mn,jm,nm]).

/* checks error for one mistype in the long command */
check_error1(Level,String_command,Correct_command) :-
    name(String_command,List_command),
    length(List_command,Length), Length > 2,
    valid(Level,Any_valid_command),
    name(Any_valid_command,Valid_list_command),
    one_difference(List_command,Valid_list_command),
    name(Correct_command,Valid_list_command),
    ask_error(Level,Correct_command).

/* checks transposing error in command */
check_error1(Level,String_command,Transposed_string_command) :-
    name(String_command,List_command),
    do_transpose(List_command,Transposed_command),
    name(Transposed_string_command,Transposed_command),
    valid(Level,Transposed_string_command),
    ask_error(Level,Transposed_string_command).

do_transpose(L,TL) :- transpose(L,TL).

transpose([X,Y|L],[Y,X|L]).
transpose([X|L],[X|L2]) :- transpose(L,L2).
check_error1(Level,Scommand,Ncommand) :- name(Scommand,Acommand),
                                         type_error_list1(Level,Ncommand,List1),
                                         member(Acommand,List1),
                                         ask_error(Level,Ncommand).

type_error_list1(Level,Command,Error) :- e(Command,Error).
e(trace,[X,114,97,99,101],[116,X,97,99,101],[116,114,X,99,101])).

/* checks error for level errors */
check_error1(Level,Scommand,Ncommand) :- level_error_list(Level,Ncommand),
                                         correct_level_error(Level,Ncommand).

/* Correct the errors in misconception of level */

```



```

correct_level_error(top,E) :- nl,write('Oops! You can issue a " '),
                             write(E), write(' " command only in the "MM" level. '),
                             nl,nl, write('Try again. '),nl.
/*
                             NO =.. [command,top,E],
                             check_with_student(O,S,D,NO),!.
*/

correct_level_error(mm,E) :- nl,write('Oops! You can issue a " '),
                             write(E), write(' " command only in the "TOP" level. '),
                             nl,nl, write('Try again. '),nl.
/*
                             NO =.. [command,mm,E],
                             check_with_student(O,S,D,NO),!.
*/

/* Ask the user if he/she meant the valid command. */
ask_error(Level,Correct_command) :- nl,
                                   write('Oops! You mean " '), write(Correct_command), write(' "? '),
                                   niceread(Answer), nl,
                                   !, name(Yes_or_no,Answer),
                                   NC =.. [command,Level,Correct_command],
                                   correct_error(Yes_or_no,Level,NC).

/* Correct the error if the answer is "yes" */
correct_error(A,Level,NC) :- affirmative(A), nl, action(Level,NC), !.

try_again(Level,NC) :- , !.

/* Correct the error if the answer is "no" */
correct_error(A,Level,NC) :- negative(A),nl,
                             !, write('Please, type the correct command. '),nl.

/* If the answer is neither "yes" nor "no" */
correct_error(A,Level,NC) :- nl, write('Please, answer in "yes" or "no". '),
                             nl, write('==> '), niceread(B), name(S,B),
                             correct_error(S,Level,NC).

/* valid commands in TOP level */

valid(top,mm).
valid(top,q).

/* valid commands in MM level */

valid(mm,answer).
valid(mm,copy).
valid(mm,delete).
valid(mm,exit).
valid(mm,expunge).
valid(mm,flag).
valid(mm,forward).
valid(mm,'headers all').
valid(mm,'headers current').
valid(mm,'headers delete').
valid(mm,'headers flag').
valid(mm,'headers unseen').

```

```

valid(nm,help).
valid(nm,jump).
valid(nm,list).
valid(nm,logout).
valid(nm,move).
valid(nm,next).
valid(nm,previous).
valid(nm,quit).
valid(nm,q).
valid(nm,read).
valid(nm,send).
valid(nm,type).
valid(nm,undelete).
valid(nm,unflag).

/* valid commands in READ mode */

valid(read,copy).
valid(read,delete).
valid(read,flag).
valid(read,forward).
valid(read,help).
valid(read,list).
valid(read,move).
valid(read,next).
valid(read,quit).
valid(read,q).
valid(read,reply).
valid(read,send).
valid(read,undelete).
valid(read,unflag).

/* valid commands in SEND mode */

valid(send,display).
valid(send,erase).
valid(send,headers).
valid(send,quit).
valid(send,q).
valid(send,send).
valid(send,type).

level_error_list(top,answer).
level_error_list(top,copy).
level_error_list(top,delete).
level_error_list(top,exit).
level_error_list(top,expunge).
level_error_list(top,flag).
level_error_list(top,forward).
level_error_list(top,'headers all').
level_error_list(top,'headers answer').
level_error_list(top,'headers delete').
level_error_list(top,'headers flag').
level_error_list(top,'headers unseen').
level_error_list(top,help).
level_error_list(top,jump).
level_error_list(top,list).
level_error_list(top,next).
level_error_list(top,previous).
level_error_list(top,read).

```

```
level_error_list(top,send).
level_error_list(top,type).
level_error_list(top,undelete).
level_error_list(mm,mm).
```

```
affirmative(yes).
affirmative(yep).
affirmative(yap).
affirmative(ya).
affirmative(ye).
affirmative(ya).
affirmative(y).
affirmative(right).
affirmative(ok).
```

```
negative(no).
negative(n).
negative(nop).
negative(never).
```

```
member(X,[X|L]).
member(X,[Y|L]) :- member(X,L).
```

```
convert_string_to_ASCII(List,Alist) :- conl(List,Tlist),reverse(Tlist,Alist).
conl([],[]).
conl([X|L],Alist) :- name(X,Ascii),append(Ascii,Alist,Nalist),conl(L,Nalist).
```

```
one_difference([X|L],[Y|L]).
one_difference([X|L1],[X|L2]) :- one_difference(L1,L2).
```

LIST OF REFERENCES

1. N. C. Rowe, *Artificial Intelligence Through Prolog*, Prentice-Hall, 1988 .
2. J. S. Brown, *Intelligent Tutoring Systems*, Academic Press, 1982 .
3. E. A. Feigenbaum , *The Handbook of Artificial Intelligence*, v. 2, William Kaufmann, 1982 .
4. D. A. Waterman, *A Guide to Expert Systems*, McGraw-Hill, 1986 .
5. Charles Pine, *The C Workshop*, Word Craft, 1987 .
6. *User's Guide to TOPS-20*, USC/Information Science Institute, Marina del Rey, Ca, April 1984 .

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information System Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curriculum Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
5.	Superintendent, Naval Postgraduate School Computer Technology Programs, Code 37 Monterey, California 93943-5000	1
6.	Professor Neil C. Rowe, Code 52Rp Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
7.	Professor Gary K. Poock, Code 55PK Naval Postgraduate School Monterey, California 93943-5000	1
8.	Kim, Sung Jin National Computerization Agency Juksun Hyundai Bldg Office 601 80 Juksundong Jongrogu, Seoul, Korea 110	1

- | | | |
|-----|---|---|
| 9. | Capt. Ok, Do Kyeong
SMC 2211 NPGS
Monterey, Ca. 93943 | 1 |
| 10. | Capt. Park, Nai Soo
SMC 1831 NPGS
Monterey, Ca. 93943 | 1 |
| 11. | Capt. Kim, Yong Joo
SMC 1017 NPGS
Monterey, Ca. 93943 | 1 |
| 12. | Capt. Park, Soon Sang
SMC 2808 NPGS
Monterey, Ca. 93943 | 1 |
| 13. | Capt. Park, Hun Keun
SMC 2997 NPGS
Monterey, Ca. 93943 | 1 |
| 14. | 1Lt. Ryoo, Moo Bong
SMC 1845 NPGS
Monterey, Ca. 93943 | 1 |
| 15. | Kim, Tae Woo
5-601 Han Yang Apt.
695 Ja Yang Dong. Sung Dong Gu
Seoul, Korea 133-190 | 8 |