

KADI SARVA VISHWAVIDYALAYA
LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH
GANDHINAGAR



**Department of
Information Technology**

Subject: Artificial Intelligence (*CT601-N*)

Laboratory Manual

Prepared By:

Samarth Mistry

19BEIT30060

6IT

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH GANDHINAGAR

DEPARTMENT OF
INFORMATION TECHNOLOGY



CERTIFICATE

Mr. Mistry Samarth Alkeshkumar of 6th IT Enrolment No.
19BEIT30060 Exam No, _____ has
satisfactorily completed his/her term work in *Artificial
Intelligence (CT601-N)* for the term ending in Dec-May-2022.

Date: _____

Subject Coordinator

Dr. Mehul Barot
HOD-IT

PRACTICAL 1

AI practical #1

PAGE:
DATE:

Aim: To study of facts, objects, Predicates
Variables, Rule and Unification in PROLOG

> PROLOG (Programming in Logic)

It is a declarative programming language like SQL which has an important role in the field of Artificial Intelligence.

The primary focus in the PROLOG is the logic which is applied to perform any task. The logic is expressed with the help of some rules & facts. It is an object oriented programming language.

The organized data has known as facts. The predefined constraints & conditions to perform any task are known as rule. The queries are executed to check if the task is accomplished properly or ~~not~~ not.

PROLOG has four parts:

(i) Editor - Editor is used to write the programming logic by defining variables, facts and objects and rules.

(ii) Dialog - Dialog is used to execute the program. Here the essential queries can be executed to check if logic is implemented.

is correct or not.

(iii) Message - This blocks shows the status of action performed in code like saving the file, compilation was successful or not, etc

(iv) Trace : Trace is used to debug the program. It is used to find any errors which are not easily traceable.

(v) Facts - Facts can be described as symbolic relationships.

Example : Bob is a student
→ student(BOB);

Factual expression is called a clause.

(vi) Object : Object is the name of element of certain type. Object could be as follows,

- char : single character
- Integer : Integer values
- Real : Floating point numbers
- String : Sequence of characters
- Symbol : character sequence of left numbers & underscores with characters or lowercase letters.
- File : Symbolic name.

ex Bob is a student
T(object)

- Relation : A relation is a name that defines the way in which a collection of objects or variables belong ~~varr~~ together.

- Predicate : It is a fact without a period at the end.

ex. car is Blue.

is(Car, Blue) is not a fact, so it is a predicate.

The elements within parenthesis are arguments to predicate, which may be object or variables.

- Variables : In prolog, variable means unknown or undefined quantity. There are three types of variables.

(i) Bound variables :

- Bound means initiated or not defined at particular time.
- A variable which is bound with a value at a particular given time.
- Value is been assigned to variable then it is called bound variable.

(ii) Free variables :

These variables which are not initialized or defined at a particular given time is called.

PAGE: / /
DATE: / /

free variable.

(iii) Anonymous variables:

- It means no variable & no value
- **UNIFICATION:** Unification is a pattern matching process. The process by which a prolog tries to match a term against the facts in an efforts to prove a goal is called unification
- Predicates unify with each other if:
 - (i) They have same relation name
 - (ii) They have some number of arguments
 - (iii) All arguments pairs unify with each other
- Rules: A Rule is an expression that shows that the truth of a particular fact depends upon one or more facts. A Rule express a relationship between facts.
- Prolog expression are composed of following truth functional symbols.

<u>English</u>	<u>Predicate</u>	<u>Prolog</u>
AND	\wedge	,
OR	\vee	;
IF	\rightarrow	::
NOT	\sim	not

- A Rule is a predicate expression that uses logical implementation (`:-`) to describe a relationship among facts.

Thus a prolog rule takes the form
left hand side :- right hand side.

This is interpreted as
left hand side \Leftarrow if right hand side.

It is restricted to a single positive literal which means it must consist of a positive atomic expression. It can't be negated and it can't contain logical connectives.

This notation is known as Horn clauses.

- In Horn clauses logic, the left hand side of clause is conclusion & must be single positive literal. The right hand side contains the premises. The Horn clause calculus is equivalent to first order predicate calculus.

Part-1: To implement a program for likes.

Use of Bound Variables:

The screenshot shows a Prolog IDE interface with the following details:

- Editor:** Contains the following code:


```
Line 14 Col 25 C:\NP1.PRO Indent Insert
domains
disease, indication=symbol

predicates
symptom(disease, indication)

clauses
symptom(chicken_pox, high_fever).
symptom(chicken_pox, chills).
symptom(flu, chills).
symptom(cold, mild_body_ache).
symptom(flu, sever_body_ache).
symptom(cold, runny_nose).
symptom(flu, runny_nose).
```
- Options Dialog:** Shows the execution trace for the query `likes(rohan, smit)`:


```
Yes
Goal: likes(rohan, smit)
No
Goal: likes(rohan, X)
No
Goal: likes(rohan, X)
X=car
1 Solution
Goal: likes(X, car)
X=rohan
1 Solution
Goal: symptom(flu, chills)
)
Yes
Goal: -
```
- Message:** Displays save and compilation messages:


```
Save C:\NP1.PRO
Save C:\NP1.PRO
Compiling C:\NP1.PRO
symptom
```
- Trace:** An empty window.
- Bottom Bar:** Shows keyboard shortcuts for file operations like Save (F2), Load (F3), and End (F10).

Use of Free Variables :

The screenshot shows a Prolog IDE interface with the following details:

- Editor:** Contains the following code:


```
Line 1 Col 1 C:\WORK.PRO Indent Insert
domains
a,b=symbol
predicates
likes(a,b)
clauses
likes(rohan,car).
```
- Options Dialog:** Shows the execution trace for the query `likes(rohan, X)`:


```
Yes
Goal: likes(rohan, smit)
No
Goal: likes(rohan, X)
No
Goal: likes(rohan, X)
X=car
1 Solution
Goal: likes(X, car)
X=rohan
1 Solution
Goal: symptom(flu, chills)
)
Yes
Goal: d
```
- Message:** Displays save and compilation messages:


```
Save C:\NP1.PRO
Compiling C:\NP1.PRO
symptom
Load C:\WORK.PRO
```
- Trace:** An empty window.
- Bottom Bar:** Shows keyboard shortcuts for various functions like Help (F1), Save (F2), and Menu (F10).

Use of Anonymous Variables:

The screenshot shows a Prolog development environment with the following interface elements:

- Menu Bar:** Files, Edit, Run, Compile, Options, Setup, Dialog.
- Editor Tab:** Line 1, Col 2, C:\NP1.PRO, Indent, Insert.
- Editor Content:**

```
domains
human1,human2=symbol

predicates
likes(human1,human2)

clauses
likes(smitesh,samarth).
likes(parth,rohan).
likes(rohan,parth).
likes(samarth,rohan).
```
- Dialog Tab:** A list of query results:

```
Yes
Goal: likes(smitesh,rohan)
No
Goal: likes(smitesh,X)
X=samarth
1 Solution
Goal: likes(X,parth)
X=rohan
1 Solution
Goal: likes(.,samarth)
Yes
Goal: likes(smitesh,.)
Yes
Goal: _
```
- Message Tab:** Compiling C:\NP1.PRO, Compilation successful.

```
Compiling C:\NP1.PRO
Compilation successful
Likes
Likes
```
- Trace Tab:** (Empty)
- Bottom Bar:** F2-Save, F3-Load, F5-Zoom, F6-Next, F8-Previous goal, Shift-F10-Resize, F10-End.

Part 2: To implement a program of Diseases and it's Symptoms.

Use of Bound Variables:

```

Line 2   Col 11   WORK.PRO   Indent   Insert
domains
diseases,symptom=symbol

predicates
relation(diseases,symptom)

clauses
relation(flu,fever).
relation(common_cold,runny_nose).
relation(migrain,head_ache).
relation(allergy,runny_nose).
relation(cholera,body_ache).

Goal: relation(flu,fever)
Yes
Goal: relation(migrain,head_ache)
Yes
Goal: relation(allergy,runny_nose)
No
Goal: relation(allergy,runny_nose)
Yes
Goal: _
```

Message

```

Compiling WORK.PRO
Compiling WORK.PRO
Compilation successful
relation
```

Trace

F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End

Use of Free Variables:

```

Line 2   Col 11   WORK.PRO   Indent   Insert
domains
diseases,symptom=symbol

predicates
relation(diseases,symptom)

clauses
relation(flu,fever).
relation(common_cold,runny_nose).
relation(migrain,head_ache).
relation(allergy,runny_nose).
relation(cholera,body_ache).

Goal: relation(allergy,runny_nose)
No
Goal: relation(allergy,runny_nose)
Yes
Goal: relation(X,runny_nose)
X=common_cold
X=allergy
Z Solutions
Goal: relation(allergy,X)
X=runny_nose
1 Solution
Goal:
```

Message

```

Compilation successful
relation
relation
relation
```

Trace

F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End

Use of Anonymous Variables:

The screenshot shows a Prolog development environment with several panes:

- Editor:** Displays the source code of a file named WORK.PRO. The code defines domains, predicates, and clauses. It includes predicates for diseases and symptoms, and clauses for various medical conditions like flu, common cold, migraine, allergy, and cholera.
- Dialog:** Shows the execution of a query. The user asks for relations involving an anonymous variable X. The system responds with two solutions: one where X is 'allergy' and another where X is 'runny_nose'. Both queries succeed.
- Message:** Shows the compilation message: "Compilation successful". It also lists the predicates defined in the file: relation, relation, relation, and relation.
- Trace:** This pane is currently empty.
- Bottom Bar:** Contains keyboard shortcuts for file operations: F2-Save, F3-Load, F5-Zoom, F6-Next, F8-Previous goal, Shift-F10-Resize, and F10-End.

```

Line 2   Col 11   WORK.PRO   Indent   Insert
domains
diseases,symptom=symbol

predicates
relation(diseases,symptom)

clauses
relation(flu,fever).
relation(common_cold,runny_nose).
relation(migrain,head_ache).
relation(allergy,runny_nose).
relation(cholera,body_ache).

Message
Compilation successful
relation
relation
relation

Trace

F2-Save  F3-Load  F5-Zoom  F6-Next  F8-Previous goal  Shift-F10-Resize  F10-End
  
```

Part 3 : Implement the following knowledge with the statement.

Knowledge:

- (a.) Burger is a Food
- (b.) Sandwich is a Food.
- (c.) Pizza is a Food.
- (d.) Sandwich is a Lunch.
- (e.) Pizza is a Dinner.

Statement: Anything is a meal, if it is a food.

The screenshot shows a Prolog development environment with the following interface elements:

- Menu Bar:** Files, Edit, Run, Compile, Options, Setup, Dialog.
- Editor Area:** Shows the Prolog code. The cursor is at Line 13, Col 14, in file C:\WORK.PRO. The code defines predicates for meal, food, lunch, and dinner.
- Output Area (Dialog Box):** Displays the execution results of the query `meal(burger)`. It shows three solutions:
 - W=burger
 - W=sandwich
 - W=pizza
- Message Area:** Shows the compilation process for C:\WORK.PRO, listing the predicates: food, lunch, and meal.
- Trace Area:** A large empty window for monitoring the execution trace.
- Keyboard Shortcuts:** F2-Save, F3-Load, F5-Zoom, F6-Next, F8-Previous goal, Shift-F10-Resize, F10-End.

```

Line 13   Col 14   C:\WORK.PRO   Indent   Insert
meal(name)
clauses
food(burger).
food(sandwich).
food(pizza).
lunch(sandwich).
lunch(pizza).
dinner(pizza).
meal(X):-food(X).

Goal: meal(burger)
Yes
Goal: meal(W)
W=burger
W=sandwich
W=pizza
3 Solutions
Goal: meal(X),lunch(X)
X=sandwich
1 Solution
Goal: meal(X),lunch(X)
X=sandwich
X=pizza
2 Solutions
Goal: -

```

Part 4: Implement the following knowledge with the statement.

Knowledge:

- (a) A studies Java
- (b) B studies Java
- (c) C studies Python
- (d) B studies DS
- (e) I teaches Java
- (f) T teaches Python
- (g) T teaches CNS
- (h) Q teaches DS

Statement: X is a professor of Y if it X teaches C and Y studies C.

The screenshot shows a Prolog development environment with the following interface elements:

- Menu Bar:** Files, Edit, Run, Compile, Options, Setup.
- Toolbar:** Editor, Dialog.
- Code Area:** Shows the following Prolog code in the Editor tab:


```
Line 1   Col 1   C:\NP2(B).PRO  Indent  Insert
domains
student,teacher,subject=symbol
predicates
studies(student,subject)
teaches(teacher,subject)
professor(teacher,student)
clauses
studies(a,java).
studies(b,java).
studies(c,python).
studies(d,ds).
teaches(i,java).
teaches(t,python).
teaches(p,cn).
teaches(q,ds).
professor(X,Y):-teaches(X,C),studies(Y,C).
```
- Output Area:** Shows the results of the query in the Dialog tab:


```
2 Solutions
Goal: meal(X);lunch(X)
X=burger
X=sandwich
X=pizza
X=sandwich
X=pizza
5 Solutions
Goal: professor(i,b)
Yes
Goal: professor(q,d)
Yes
Goal: professor(p,c)
No
Goal:
```
- Trace Window:** Located at the bottom right, showing the trace of the query execution.
- Bottom Bar:** F2-Save, F3-Load, F6-Switch, F9-Compile, Alt-X-Exit.

PRACTICAL :2

AIM: Write a program to implement the “cut” and “fail” predicate in PROLOG.

Part 1: Implement program to find Minimum and Maximum number without using cut and fail predicates in prolog.

The screenshot shows a Prolog IDE interface. The menu bar includes Files, Edit, Run, Compile, Options, and Setup. A sub-menu 'Editor' is open, showing the current file path: Line 10 Col 1 C:\NP2(1).PRO. The main window displays the following Prolog code:

```

domains
a,b=integer

predicates
max(a,b)
min(a,b)

clauses
max(A,B):-A>B,write(A," is Max."),nl,write(B," is Ma
min(A,B):-B>A,write(A," is Min."),nl,write(B," is Mi

```

To the right of the code, the 'Dialog' window shows the execution of the predicates:

```

Goal: max(4,5)
5 is Max. Yes
Goal: min(4,5)
4 is Min.
Yes
Goal: _

```

Below the code editor, the 'Message' window shows the compilation status:

```

Compiling C:\NP2(1).PRO

```

Part 2: Write a program to find Minimum and Maximum using cut and fail predicates

The screenshot shows a Prolog IDE interface. The menu bar includes Files, Edit, Run, Compile, Options, and Setup. A sub-menu 'Editor' is open, showing the current file path: Line 12 Col 38 C:\NP2(2).PRO. The main window displays the following Prolog code:

```

domains
a,b=integer

predicates
max(a,b)
min(a,b)

clauses
max(A,B):-A>=B,write(A," is Max."),nl,! ,fail.
max(A,B):-A<B,write(B," is Max."),nl.
min(A,B):-A<B,write(A," is Min."),nl,! ,fail.
min(A,B):-B<A,write(B," is Min."),nl.

```

To the right of the code, the 'Dialog' window shows the execution of the predicates:

```

Goal: max(4,5)
5 is Max. Yes
Goal: min(4,5)
4 is Min.
Yes
Goal: max(9,10)
10 is Max.
Yes
Goal: min(9,10)
9 is Min.
No
Goal:

```

Below the code editor, the 'Message' window shows the results of the queries:

```

max
min
Save C:\NP2(2).PRO
Save C:\NP2(2).PRO

```

At the bottom, a toolbar provides keyboard shortcuts: F2-Save, F3-Load, F6-Switch, F9-Compile, Alt-X-Exit.

PRACTICAL : 3

AIM: Write a program to implement arithmetic operators, simple input/output and compound goals in PROLOG.

Program 1: Program to implement arithmetic operators.

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program... — X

Files Edit Run Compile Options Setup Editor Dialog

```

Line 1   Col 1   C:\PRA-3(1).PRO Indent Inse
domains
A,B=integer

predicates
add(A,B)
sub(A,B)
mul(A,B)
div(A,B)

clauses
add(A,B):-
    Z=A+B,
    write(Z),nl.
sub(A,B):-

```

Goal: add(10,20)
30
Yes
Goal: add(-10,-20)
-30
Yes
Goal: sub(20,10)
10
Yes
Goal: sub(20,-10)
30
Yes
Goal: _

Message Trace

```

Compiling C:\PRA-3(1).PRO
Compilation successful
add
sub

```

F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program... — X

Files Edit Run Compile Options Setup Editor Dialog

```

Line 22   Col 21   C:\PRA-3(1).PRO Indent Inse
clauses
add(A,B):-
    Z=A+B,
    write(Z),nl.
sub(A,B):-
    Z=A-B,
    write(Z),nl.
mul(A,B):-
    Z=A*B,
    write(Z),nl.
div(A,B):-
    Z=A/B,
    write(Z),nl.

```

30
Yes
Goal: mul(20,10)
200
Yes
Goal: mul(-10,20)
-200
Yes
Goal: div(10,20)
0.5
Yes
Goal: div(20,-10)
-2
Yes
Goal: _

Message Trace

```

Compiling C:\PRA-3(1).PRO
Compilation successful
mul
div

```

F2-Save F3-Load F5-Zoom F6-Next F8-Previous goal Shift-F10-Resize F10-End

Program 2 : Program to implement tower of hanoi .

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program... — X

Editor

```

Line 1   Col 1   C:\NPRA-3(2).PRO  Indent  Inse
domains
loc=right;middle;left

predicates
hanoi(integer)
move(integer,loc,loc,loc)
inform(loc,loc)

clauses
hanoi(N):-
    move(N, left, middle, right).
move(1,A,_C):-
    inform(A,C),!.
move(N,A,B,C):-

```

Dialog

```

Move a disk from loc1 to
loc2
Goal:

```

Message

```

hanoi
move
inform
Save C:\NPRA-3(2).PRO

```

Trace

F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu

DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program... — X

Editor

```

Line 18  Col 9   C:\NPRA-3(2).PRO  Indent  Inse
hanoi(integer)
move(integer,loc,loc,loc)
inform(loc,loc)

clauses
hanoi(N):-
    move(N, left, middle, right).
move(1,A,_C):-
    inform(A,C),!.
move(N,A,B,C):-
    N1=N-1,move(N1,A,C,B),
    inform(A,C),move(N1,B,A,C).
inform(Loc1,Loc2):-nl,
    write("Move a disk from ",loc1, " to ",loc2)

```

Dialog

```

Move a disk from loc1 to
loc2
Goal:

```

Message

```

hanoi
move
inform
Save C:\NPRA-3(2).PRO

```

Trace

F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu

Program 3 : Program to find grade .

The screenshot shows the DOSBox environment with the following details:

- Menu Bar:** Files, Edit, Run, Compile, Options, Setup, Dialog.
- Editor Area:** Shows a Prolog program named C:\NPR A-3(3).PRO. It defines domains, predicates grade(MARK), and clauses for grades A, B, and C based on MARK values.
- Dialog Box:** Displays the execution trace for the goal grade(10), showing steps for C, Yes, Goal: grade(41), B, Yes, Goal: grade(95), No, Goal: grade(65), A, Yes, and Goal: _.
- Message Area:** Shows compilation messages: Save C:\NPR A-3(3).PRO, Compiling C:\NPR A-3(3).PRO, Compilation successful, grade.
- Trace Area:** Empty.
- Bottom Bar:** F2-Save, F3-Load, F5-Zoom, F6-Next, F8-Previous goal, Shift-F10-Resize, F10-End.

Program 4 : Program to print the values between the given range.

The screenshot shows the DOSBox environment with the following details:

- Menu Bar:** Files, Edit, Run, Compile, Options, Setup, Dialog.
- Editor Area:** Shows a Prolog program named C:\NPR A-3(4).PRO. It defines domains A,B,C=integer, predicates range(integer,integer,integer), and clauses for range(Out,Low,High) using Newlow=Low+1 and Newlow<=High.
- Dialog Box:** Displays the execution trace for the goal range(A,1,10), listing values from A=1 to A=10, followed by 10 Solutions and Goal: _.
- Message Area:** Shows compilation messages: Save C:\NPR A-3(4).PRO, Compiling C:\NPR A-3(4).PRO, Compilation successful, range.
- Trace Area:** Empty.
- Bottom Bar:** F2-Save, F3-Load, F5-Zoom, F6-Next, F8-Previous goal, Shift-F10-Resize, F10-End.

PRACTICAL 4

AIM: Write a program to implement recursion in PROLOG. Program to implement factorial.

The screenshot shows a Prolog development environment with the following details:

- Editor:** Shows the Prolog code for calculating factorial. It includes a base case for 0 and a recursive case for n > 0, defining N1 as N-1 and F1 as the factorial of N1, then multiplying N by F1.
- Dialog:** Displays the execution trace:
 - Goal: larger(5,7,X). X=7
 - Goal: factorial(5,X). X=120
 - Goal: factorial(10,X). X=24320
 - Goal: factorial(5,X). X=120
 - Goal: factorial(10,X). X=24320
 - Goal: factorial(10,X). X=24320
 - Goal:
- Message:** A message box containing:
 - Press the SPACE bar
 - Save C:\FACTORYL.PRO
 - Compiling C:\FACTORYL.PRO
 - factorial
- Trace:** An empty window for monitoring the execution process.
- Bottom Bar:** Contains keyboard shortcuts: F2-Save, F3-Load, F5-Zoom, F6-Next, F8-Previous goal, Shift-F10-Resize, F10-End.

PRACTICAL 5

AIM: Write a program to implement Lists in PROLOG.

Part 1: Implementing program to add an element to the List in PROLOG.

The screenshot shows a Prolog development environment with the following details:

- Editor:** Contains the source code for a Prolog program named C:\P5_P1.PRO. The code defines domains for numbers and lists, predicates for adding elements to lists, and a clause for addelement.
- Dialog:** Shows the execution of several goals:
 - Goal: factorial(6,X)
X=720
 - Goal: addelement(2,[10,20,30],List)
List=[2,10,20,30]
 - 1 Solution
 - Goal: addelement(0,[10,20,30],List)
List=[0,10,20,30]
 - 1 Solution
 - Goal: addelement(100,[10,20,30],List)
List=[100,10,20,30]
 - 1 Solution
 - Goal: S_
- Message:** Displays repeated messages about saving the file to C:\P5_P1.PRO.
- Trace:** An empty window for monitoring the execution trace.
- Bottom Bar:** Contains keyboard shortcuts for file operations (F2-Save, F3-Load, etc.) and other functions.

Part 2: The PROLOG program defines the relation last(item,list) so that the item is the last element of the list using concatenation.

The screenshot shows a PROLOG development environment with the following interface elements:

- Menu Bar:** Files, Edit, Run, Compile, Options, Setup, Dialog.
- Editor Area:**

```

Line 1   Col 2      C:\NP5_P2.PRO  Indent  Insert
Sdomains
x=integer
list=integer*

predicates
concatenate(list,list,list)
last(x,list)

clauses

concatenate([],List,List).
concatenate([X|List1],List2,[X|List3]):- 
    concatenate(List1,List2,List3).

```
- Message Area:** Shows the compilation process:


```

Save C:\NP5_P2.PRO
Compiling C:\NP5_P2.PRO
concatenate
last
    
```
- Trace Area:** Empty.
- Bottom Bar:** F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu

This screenshot is similar to the one above, but it includes additional clauses for the last/2 predicate:

```

Line 18  Col 2      C:\NP5_P2.PRO  Indent  Insert
concatenate(list,list,list)
last(x,list)

clauses

concatenate([],List,List).
concatenate([X|List1],List2,[X|List3]):- 
    concatenate(List1,List2,List3).

last(X,List):-
    concatenate(List,[X],List1),
    write("The List is : ",List1),nl.

```

The message area shows the addition of the new clause:

```

Save C:\NP5_P2.PRO
Compiling C:\NP5_P2.PRO
concatenate
last
S
    
```

PRACTICAL : 6

Program (1): Code for Prolog program to check whether a given word is a palindrome or not in Artificial Intelligence.

domains

x = char

l = char*

predicates

palindrome(l)

reverse(l,l)

concatenate(l,l,l)

clauses

concatenate([],List,List).

concatenate([X|List1],List2,[X|List3]) :-
concatenate(List1,List2,List3).

reverse([],[]).

reverse([X|Tail],List) :-

reverse(Tail,Tail1),

concatenate(Tail1,[X],List).

palindrome(List):-

reverse(List,List).

OUTPUT :

```
Goal: palindrome(['m'])
Yes
Goal: palindrome(['h','e
','l'])
No
Goal: _
```

Program (2): Code for Prolog program to check whether a given list is palindrome or not in Artificial Intelligence

Domains

list=symbol*

predicates

palin(list)

findrev(list,list,list)

compare(list,list)

clauses

palin(List1):-

 findrev(List1,[],List2),

 compare(List1,List2).

findrev([],List1,List1).

findrev([X|Tail],List1,List2):-

 findrev(Tail,[X|List1],List2).

 compare([],[]):-

 write("\nList is Palindrome").

compare([X|List1],[X|List2]):-

 compare(List1,List2).

compare([X|List1],[Y|List2]):-

 write("\nList is not Palindrome").

OUTPUT :

```

Dialog
Goal: palin([m,i,t]).  

List is not Palindrome  

Yes  

Goal: palin([n,y,n]).  

List is Palindrome  

Yes  

Goal:

```

Program (3): Code for Prolog program to compare characters, strings and also reverse string in Artificial Intelligence

```

domains
strlist = string*
predicates
start
    createlist(integer,strlist,strlist,strlist)
    strcmp(string,string)
    charcmp(char,string)
    reverse(strlist,strlist,strlist)
goal
    clearwindow,
    start.
clauses
start:
-
createlist(3,[],Newlist,List1), reverse(List1,[],List2),
List2 = [Str1 | Tail], Tail = [Str2|Tail1],
Tail1=[Str3|Str4],
write("cmp string1 and string2"),nl, strcmp(Str1,Str2),
write("cmp string1 and string3"),nl, strcmp(Str1,Str3),
write("cmp string2 and string3"),nl, strcmp(Str2,Str3).
createlist(Num,Oldlist,Newlist,List1):
    Num > 0,
    write("Enter any string="), readIn(Str),
    Newlist = [Str | Oldlist],
    NN = Num -1,
    createlist(NN,Newlist,List2,List1).
createlist(_,Oldlist,_,List1):
    List1 = Oldlist.
strcmp(Str1,Str2):
    frontchar(Str1,Ch1,Rest1), charcmp(Ch1,Str2),
    Rest1<>"",strcmp(Rest1,Str2).
strcmp(Str1,Str2).
charcmp(Ch1,Str2):
frontchar(Str2,Ch2,Rest2),
Ch1<>Ch2,Rest2<>"",
charcmp(Ch1,Rest2).
charcmp(Ch1,Str2):
    frontchar(Str2,Ch2,Rest2),
    Ch1=Ch2,
    write("char=",Ch2),nl.
charcmp(Ch1,Str2).
reverse([],Inputlist,Inputlist).
```

```
reverse([Head | Tail],List1,List2):-  
reverse(Tail,[Head | List1],List2).
```

OUTPUT :

```
----- Dialog -----  
Enter any string =mit  
Enter any string =mi  
cmp string1 and string2  
char=m  
char=i  
char=t  
cmp string1 and string3  
char=m  
char=i  
cmp string2 and string3  
char=m  
char=i
```

PRACTICAL : 7 .

Program (1): Code for Prolog program for family hierarchy in Artificial Intelligence.

Predicates

```
male(symbol).
female(symbol).
father(symbol,symbol).
husband(symbol,symbol).
brother(symbol,symbol).
sister(symbol,symbol).
listbrothers(symbol).
listsisters(symbol).
mother(symbol,symbol).
grandfather(symbol).
grandmother(symbol).
uncle(symbol).
aunt(symbol).
cousin(symbol).
listgrandsons(symbol).
listgranddaughters(symbol).
printmenu.
action(integer).
repeat.
```

clauses

```
male(dashrath).
male(ram).
male(laxman).
male(bharat).
male(luv).
male(kush).
male(son_of_laxman).
female(kaushalya).
female(sita).
female(urmila).
female(daughter_of_dashrath).
father(dashrath,ram).
father(dashrath,laxman).
father(dashrath,bharat).
father(ram,luv).
father(ram,kush).
father(laxman,son_of_laxman).
father(dashrath,daughter_of_dashrath).
husband(dashrath,kaushalya).
husband(ram,sita).
```

husband(laxman,urmila).

mother(X,Y):-

```
    husband(Z,X),
    father(Z,Y).
```

brother(X,Y):-

```
    father(Z,X),
    father(Z,Y),
    X<>Y,
    male(X).
```

sister(X,Y):-

```
    father(Z,X),
    father(Z,Y),
    X<>Y,
    female(X).
```

listbrothers(X) :-

```
    brother(Z,X),
    write(Z).
```

listsisters(X):-

```
    sister(Z,X),
    write(Z).
```

grandfather(X):-

```
    father(Y,Z),
    father(Z,X),
    write(Y, " is the grandfather of ",X,"\\n").
```

grandmother(X):-

```
    husband(Z,X),
    father(Z,V),
    father(V,Y),
    write(Y, " is the grandmother of ",X,"\\n").
```

listgrandsons(X):-

```
    father(X,Z),
    father(Z,Y),
    male(Y),
    write(Y,"\\n"),
    Fail.
```

listgrandsons(X):-

```
    husband(Y,X),
    father(Y,V),
    father(V,Z),
    male(Z),
    write(Z,"\\n"), fail.
```

listgranddaughters(X):-

```
    father(X,Z),
    father(Z,Y),
```

```

female(Y),
write(Y,"\\n"), fail.

listgranddaughters(X):-
    husband(Y,X), father(Y,V),
    father(V,Z),
    female(Z),
    write(Z,"\\n"), fail.

uncle(X):-
    brother(Z,Y),
    father(Z,X),
    male(Y),
    write(Y,"\\n"),
    fail.

aunt(X):-
    husband(Z,Y),
    brother(Z,V),
    father(V,X),
    write(Y,"\\n"),
    fail.

cousin(X):-
    father(Z,X),
    father(V,Y),
    Z<>V,
    brother(V,Z),
    write(Y,"\\n").
    repeat.

repeat:- repeat.

action(1):-
    write("\nEnter name of person whose father is to be found: "),
    readln(X),
    write("\\n"),
    write("Father of ",X,"is:"), 
    father(Z,X),
    write(Z,"\\n"),
    fail.

action(2):-
    write("\nEnter name of person whose mother is to be found: "), readln(X),
    write("\\n"),
    write("Mother of ",X," is:"), 
    mother(Z,X), write(Z,"\\n"), fail.

action(3):- write("\nEnter name of person whose brothers are to be found: "),
    readln(X),
    write("\\n"),
    write("Brothers of ",X,"are:\\n"),

```

```
listbrothers(X),
write("\n"),
fail.

action(4):-
    write("\nEnter name of person whose sisters are to be found: "),
    readln(X),
    write("\n"),
    write("Sisters of ",X,"are:\n"),
    listsisters(X),
    write("\n"),
    fail.

action(5):-
    write("\nEnter name of person whose grandsons are to be found: "),
    readln(X),
    write("\n"),
    write("Grandsons of ",X,"are:\n"),
    listgrandsons(X),
    write("\n"),
    fail.

action(6):-
    write("\nEnter name of person whose granddaughters are to be found: "),
    readln(X),
    write("\n"),
    write("Granddaughters of ",X,"are:\n"),
    listgranddaughters(X),
    write("\n"),
    fail.

action(7):-
    write("\nEnter name of person whose uncles are to be found: "),
    readln(X),
    write("\n"),
    write("Uncles of ",X,"are:\n"),
    uncle(X),
    write("\n"),
    fail.

action(8):-
    write("\nEnter name of person whose aunties are to be found: "),
    readln(X),
    write("\n"),
    write("Aunties of ",X," are:\n"),
    aunt(X),
    write("\n"),
    fail.
```

```
action(9):-
```

```
    write("\nEnter name of person whose cousins are to be found: "),
    readIn(X),
    write("\n"),
    write("Cousins of ",X,"are:\n"),
    cousin(X),
    write("\n"),
    fail.
```

```
action(0).printmenu:-
```

```
repeat,
write("\n1. Display Father of?\n"),
write("2. Display Mother of?\n"),
write("3. List all brothers of?\n"),
write("4. List all sisters of?\n"),
write("5. List all grandson of?\n"),
write("6. List all granddaughterof?\n"),
write("7. List all unclesof?\n"),
write("8. List all auntyof?\n"),
write("9. list all cousinsof?\n"),
write("0. exit\n"),
write("Enter your choice :"),
readInt(Choice),
action(Choice),
write("\n"), repeat.
```

```
goal
```

```
makewindow(1,2,3,"Family
Tree",0,0,25,80), printmenu.
```

OUTPUT :

```
+ .....-Family Tree .....
|  
|1. Display Father of?  
|2. Display Mother of?  
|3. List all brothers of?  
|4. List all sisters of?  
|5. List all grandson of?  
|6. List all granddaughter of?  
|7. List all uncles of?  
|8. List all aunty of?  
|9. list all cousins of?  
|0. exit  
|Enter your choice : 1  
  
|Enter name of person whose father is to be found: ram  
  
|Father of ram is: dashrath  
  
|1. Display Father of?  
|2. Display Mother of?  
|3. List all brothers of?  
|4. List all sisters of?  
|5. List all grandson of?  
|6. List all granddaughter of?  
|7. List all uncles of?  
|8. List all aunty of?  
|9. list all cousins of?  
|0. exit  
|Enter your choice : 3  
  
|Enter name of person whose brothers are to be found: ram  
  
|Brothers of ram are:  
|laxman  
|bharat  
  
|1. Display Father of?  
|2. Display Mother of?  
|3. List all brothers of?  
|4. List all sisters of?  
|5. List all grandson of?  
|6. List all granddaughter of?  
|7. List all uncles of?  
|8. List all aunty of?  
|9. list all cousins of?
```

```
|0. exit
|Enter your choice : 5
|
|Enter name of person whose grandsons are to be found: dashrath
|
|Grandsons of dashrath are:
|luv
|kush
|son_of_laxman
|
|1. Display Father of?
|2. Display Mother of?
|3. List all brothers of?
|4. List all sisters of?
|5. List all grandson of?
|6. List all granddaughter of?
|7. List all uncles of?
|8. List all aunty of?
|9. list all cousins of?
|0. exit
|Enter your choice : 7
|
|Enter name of person whose uncles are to be found : kus
|
|Uncles of kush are:
|laxman
|bharat
|
|1. Display Father of?
|2. Display Mother of?
|3. List all brothers of?
|4. List all sisters of?
|5. List all grandson of?
|6. List all granddaughter of?
|7. List all uncles of?
|8. List all aunty of?
|9. list all cousins of?
|0. exit
|Enter your choice :
|
|Press the SPACE bar
+-----
```

PRACTICAL : 8

Program : Write a program to implement BFS (for AI search problem).
domains

X, H, N, ND=symbol
 P, L, T, Z, Z1, L1, L2, L3, PS, NP, ST, SOL=symbol*

Predicates

solve(L, L) member(X,L) extend(L, L) conc(X, L, L) breadthfirst(L, L) goal(X)

clauses

solve(start, solution):-/*solution is a state from start to a goal*/ breadthfirst ([[start]], solution).

breadthfirst([[node|path]| _] ,[node|path]):- /*solution is an extension to a goal*/
 /*of one of path*/

goal(node).

breadthfirst([path|paths], solution):- extend(path,newpaths), conc(paths,newpaths,path1),
 breadthfirst(path1,solution).

extend([node|path],newpaths):- bagof([newnode, node|path],(s(node,
 newnode),notmember(newnode,[node|path])), newpaths),!. extend(path, []).

conc([], L, L).

conc([X|L1], L2, [X|L3]):- conc(L1, L2, L3).

member(X, [X|T]).

member(X,

[H|T]):-

member(X, T).

OUTPUT :

```
goal: solve([a, e],S)
L= ["a", "b", "c", "d", "e"]

goal: solve([a, h],S)
L= ["a", "b", "c", "d", "e", "f", "g", "h"]
```

PRACTICAL : 9

Program : Write a program to implement DFS (for 8 puzzle problem)

domains

H=integer T=integer*

predicates

```
safe(T)
solution(T)
permutation(T,T)
del(H,T,T)
noattack(H,T,H)
```

clauses

```
del(I,[I|L],L). /*to take a position from the permutation oflist*/
```

```
del(I,[F|L],[F|L1]):-
```

```
del(I,L,L1).
```

```
permutation([],[]). /*to find the possiblepositions*/
```

```
permutation([H|T],PL):-
```

```
    permutation(T,PT),\ del(H,PL,PT).
```

```
solution(Q):- /*final solution is stored inQ*/
```

```
    permutation([1,2,3,4,5,6,7,8],Q),
```

```
    safe(Q).
```

```
safe([]). /*Q is safe such that no queens attack eachother*/
```

```
safe([Q|others]):-
```

```
    safe(others),
```

```
    noattack(Q,others,1).
```

```
noattack(_,[],_) /*to find if the queens are in same row, column or
diagonal*/
```

```
noattack(Y,[Y1|Ydist],Xdist):-
```

```
Y1-Y>Xdist,
```

```
Y-Y1<>Xdist,
```

```
dist1=Xdist,
```

```
noattack(Y,Ydist,dist1).
```

OUTPUT:-

goal:-solution(Q). Q=[“3”,“8”,“4”,“7”,“1”,“6”,2”,“5”]

PRACTICAL : 10

Program : Write a program to implement A* Algorithm.

```
%%%  
%%%  
%%% Nodes have form S#D#F#A  
%%% where S describes the state or configuration  
%%%D is the depth of the node  
%%%F is the evaluation function value  
%%%A is the ancestor list for the node  
:- op(400,yfx,'#'). /* Node builder notation*/  
solve(State,Soln) :- f_function(State,0,F),  
search([State#0#F#[[]],S), reverse(S,Soln).  
f_function(State,D,F) :- h_function(State,H),  
F is D + H.  
search([State#_#_#Soln|_], Soln) :-  
goal(State). search([B|R],S) :-  
expand(B,Children),  
insert_all(Children,R,Ope  
n), search(Open,S).  
insert_all([F|R],Open1,Open3) :- insert  
(F,Open1,Open2),  
insert_all(R,Open2,Open3).  
insert_all([],Open,Open).  
insert(B,Open,Open) :- repeat_node(B,Open), !  
. insert(B,[C|R],[B,C|R]) :- cheaper(B,C), ! .  
insert(B,[B1|R],[B1|S]) :- insert(B,R,S),  
!. insert(B,[],[B]).  
repeat_node(P#_#_#, [P#_#_#_|_]).  
cheaper(_#_#F1#_, _#_#F2#_) :- F1 <  
F2.  
expand(State#D#_#S,All_My_Childr  
en) :-  
bagof(Child#D1#F#[Move|S],  
(D1 is D+1,  
move(State,Child,Mov  
e),  
f_function(Child,D1,F))  
, All_My_Children).
```

PRACTICAL : 11

Program : Write a program to solve travelling salesman problem using Prolog.

domains

```
town = symbol
distance = integer
```

predicates

```
Nondeterm road(town,town,distance)
nondeterm route(town,town,distance)
```

clauses

```
road("tampa","houston",200).
road("gordon","tampa",300).
road("houston","gordon",100).
road("houston","kansas_city",120).
road("gordon","kansas_city",130).
route(Town1,Town2,Distance):-
    road(Town1,Town2,Distance).
route(Town1,Town2,Distance):-
    road(Town1,X,Dist1),
    route(X,Town2,Dist2),
    Distance=Dist1+Dist2, !.
```

OUTPUT :

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: ... — X

Files Edit Run Compile Options Setup Editor Dialog

Line 1 Col 1 C:\PRR11.PRO Indent Insert

```
domains
town = symbol
distance = integer

predicates
nondeterm road(town,town,distance)
nondeterm route(town,town,distance)
clauses

road("tampa","houston",200).
road("gordon","tampa",300).
road("houston","gordon",100).
road("houston","kansas_city",120).
```

Goal: route("tampa","kansas_city",X), write("Dis",X), nl.
Dis320
X=320
1 Solution
Goal:

Message Trace

Compiling C:\PRR11.PRO

F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu

PRACTICAL : 12

Program : Study of dynamic database in PROLOG.

Predicates

```
reading
writing
delete
find(integer)
startup(integer)
```

Database

```
unsortedDatabase(string,integer)
sortedDatabase(string)
```

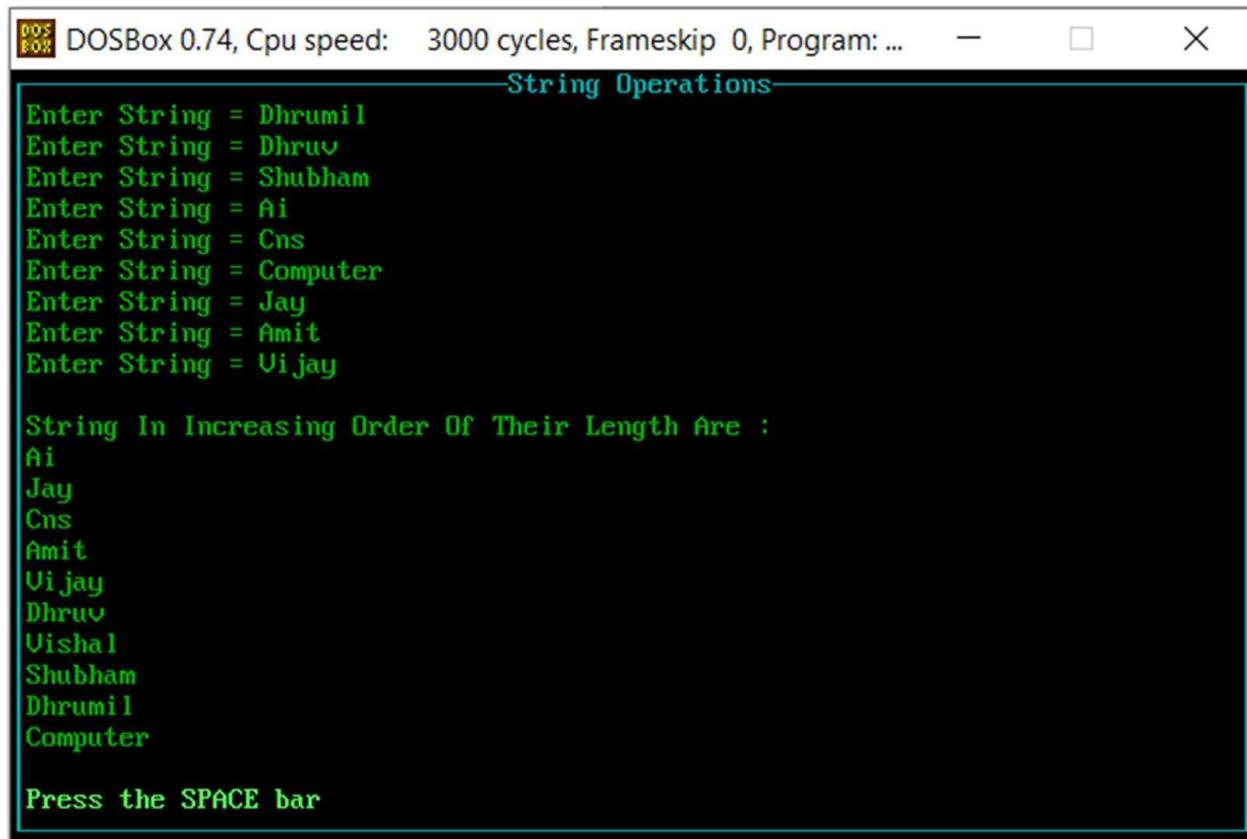
Clauses

```
startup(0).
startup(Num):- write("Enter String = "), readIn(Name), str_len(Name,Len), asserta(unsortedDatabase(Name,Len)), TempNum = Num - 1, startup(TempNum).
writing:- sortedDatabase(Name), write(Name), nl, fail.
writing.
find(Index):- unsortedDatabase(Name,Index), assertz(sortedDatabase(Name)), retract(unsortedDatabase(Name,Index)), find(Index).
find(Index):- Index = 255.
find(Index):- TempIndex = Index + 1, find(TempIndex).
reading:- NumRead = 10, startup(NumRead).
delete :- retract(sortedDatabase(_)), fail.
delete.
```

Goal

```
Clearwindow,
```

```
makewindow(1,2,3,"String Operations",0,0,25,80),
reading,!,
find(1),
write("\nString In Increasing Order Of Their Length Are :\n"),
writing,
Delete.
```

OUTPUT :

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: ... — X

String Operations

```
Enter String = Dhrumil
Enter String = Dhruv
Enter String = Shubham
Enter String = Ai
Enter String = Cns
Enter String = Computer
Enter String = Jay
Enter String = Amit
Enter String = Vijay

String In Increasing Order Of Their Length Are :
Ai
Jay
Cns
Amit
Vijay
Dhruv
Uishal
Shubham
Dhrumil
Computer

Press the SPACE bar
```