Workbook

Artificial Intelligence & Expert System (CT-361)

Name:	**M. Hasham Khalid**
Roll No:	**SE-079**
Batch:	**2018**
Year:	**Third Year**
Department:	**Software Engineering**

Table of Contents

Lab # 1

Object:

Introduction to PROLOG.

Theory:

Introduction.

Prolog was invented in the early seventies at the University of Marseille. Prolog stands for PROgramming in LOGic. It is a logic language that is particularly used by programs that use non-numeric objects. For this reason it is a frequently used language in Artificial Intelligence where manipulation of symbols is a common task. Prolog differs from the most common programming languages because it is a declarative language. Traditional programming languages are said to be procedural and the programmer must specify the details of how to solve the problem. This means that the programmer specifies how to solve a problem. In declarative languages the programmers only specify the goal to be achieved and the Prolog system works out how to achieve it. Prolog's fast incremental development cycle and rapid prototyping capabilities have encouraged the use of it as a tool for solving AI problems. Its features include interface to other languages and database products and more recently support for object oriented and constraint based programming has also been introduced.

Prolog with Object Oriented Extensions.

Object oriented extensions to Prolog have increased the attractiveness and expressive power of the language and has provided the Prolog users with a powerful and flexible object oriented development language. Constrains replace Prolog's usual pattern matching mechanism with a more general operation called constraint satisfaction. Constrains are powerful way to reduce the size of search space and increase the efficiency of the scheduler.

Applications of Prolog.

- Intelligent data base retrieval.
- Natural language understating.
- Expert systems.
- Machine learning.
- Problem solving.
- Automated reasoning.

Relations in Prolog.

Prolog programs specify relationships among objects and properties of objects. When it is

said that "Ali has a Car", an ownership relationship is being declared between two objects: Ali and the Car, and when it is asked "Does Ali own a Car?" then we are trying to find out about the relationship. Relationships can also depict rules such as two people are brothers if they both are male and they have same parents. A rule allows to find out about a relationship even if the relationship in not explicitly stated as a fact. Care must be taken when phrasing things. Following will be better: A and B both are brothers if A and B both are male and they have same mother and they have same father and A is not the same as B.

What is a Prolog program?

Programming in Prolog is very different from programming in a traditional procedural language like Pascal. In Prolog you don't say how the program will work. Prolog can be separated in two parts:

The Program.

The program, sometimes called Database is a text file (*.pl) that contain the facts and rules that will be used by the user of the program. It contains all the relations that make this program.

Query Mode.

When you launch a program you are in query mode. This mode is represented by the sign ? - at the beginning of the line. In query mode you ask questions about relations described in the program.

Loading a program.

First you have to launch your Prolog compiler. When Prolog is launched the ?- should appear meaning you are in query mode. The manner to launch a program depends of your compiler. We can load a program by typing the command consult[filename]. When you have done this you can use all the facts and rules that are contained in the program.

Exercise:

Q: What is the difference between a procedural and declarative language?

A procedural language is a kind of programming language where the programmer has to define all the steps of the procedure and what should be done in order to complete the task that is desired. Whereas, in a declarative language only the goal is defined and the rest of the details are handled by the language itself.

Q: Relationships can also depict rules. Give 2 examples (other than those specified in Lab-1).

- Two people are sisters, if they both are female and if they have the same parents.
- If it's raining and you go outside without an umbrella then you will get wet.

Q: Write 5 examples of ownership relationships.

- Amna owns a house.
- Aslam has 5 pencils.
- Ikram has a computer science degree.
- Ebad has 3 friends.
- Saleem owns a bike.

Lab # 2

Object:

Facts, Rules and Queries.

Theory:

Introduction.

In Prolog program facts are declared describing explicit relationships between objects and their properties such as Sara likes ice-cream, Hair is black, NED is a university, Tom is a cat, Shahzad teaches Harris.

Rules are declared defining implicit relationships between objects such as brother relationship and/or rules define implicit object properties i.e. A is a child of B if B is the parent of A **.**

The system can then be used to generate queries by asking questions about relationships and/or about the object properties such as does Sara like ice-cream? , Asad is the parent of whom?

Facts.

Facts are properties of objects or relationship between objects such as Zaheer has phone number 12345678. It will be written in prolog as: **phoneno(zaheer, 12345678)**.

It should be noted that:

- Names of properties/relationships begin with lowercase letters.
- The relationship name appears as the first term.
- Objects appear as comma separated arguments inside parentheses.
- A period "." must terminate the fact.
- Objects also begin with lowercase letters. They can also begin with digits and can be
- strings enclosed within quotes.
- phoneno(zaheer,12345678) is also called a predicate or clause.

Example

Person X teaches course Y.

teaches (X, Y).

teaches (sana, crs01).

teaches (amir, crs02).

Student X studies course Y.

studies (X, Y).

studies (samia, crs01).

studies (sadia, crs02).

Together these facts will form Prolog's database also called **Knowledge Base.**

Rules.

Consider the following case which produces a general rule:

Teacher will guide a student if that student is enrolled in the course which that teacher teaches.

In Prolog this will be written as:

guide(Teacher, Student) :- teaches(Teacher, Courseid),studies(Student, Courseid).

Facts are unit clauses whereas rules are non-unit clauses. Variable name will start with a capital letter.

Syntax of a Clause.

:- means **"if"** or **"is implied by"**, also called **neck symbol**. The left hand side of the neck is called the **head** and the right hand side is called the **body**. The comma stands for and also called **conjunction** and semicolon stands for or also called disjunction.

Goal or Query.

Queries are based on facts and rules. Questions can be asked based on the stored information.

Queries are terminated by full stop. To answer a query Prolog consults its database to see if it is a known fact or not.

?- teaches (sana, crs01).

Yes.

?- teaches (sana, C).

C=crs01.

Yes

If answer is Yes/True the query succeeded else if the answer is No/False then query failed.

A program of prolog consists of clauses, which are of three types: facts, rules and questions.

A procedure is a set of clauses about the same relation**.**

Example.

P:-Q:R. can be written as P:-Q. P:-R.

P:-Q,R;S,T,U. can be written as P:-(Q,R);(S,T,U). OR P:- Q,R. P:-S,T,U.

Exercise:

Enter the above program into Prolog and execute the queries shown below:

Facts & Rules

ring(Person, Number) :- location(Person, Place), phone_number(Place, Number).

location(Person, Place) :- at(Person, Place).

location(Person,Place) :- visiting(Person, Someone), location(Someone, Place).

phone_number(rm303g, 5767).

phone_number(rm303a, 5949).

at(dr_jones, rm303g).

at(dr_mike, rm303a).

visiting(dr_mike, dr_jones).

Queries.

?- location(dr_bottaci, Pl).

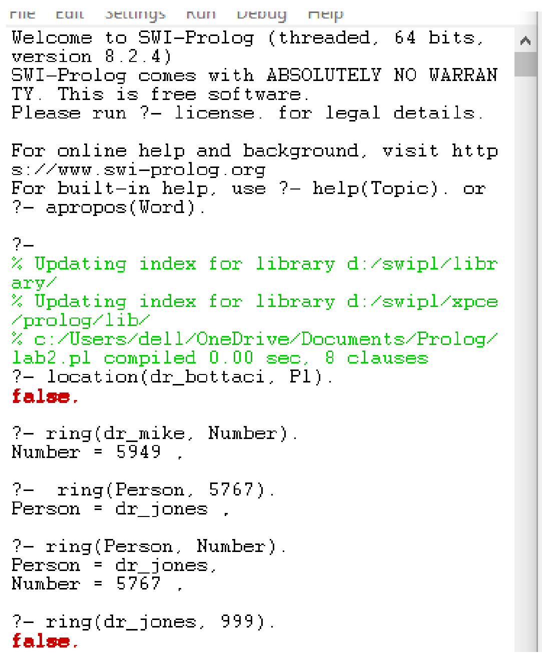
?- ring(dr_mike, Number).

?- ring(Person, 5767).

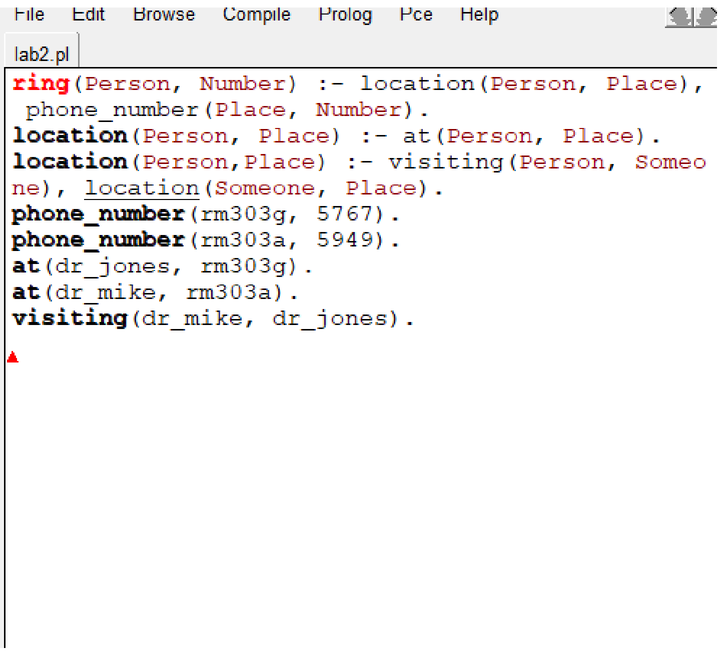
?- ring(Person, Number).

?- ring(dr_jones, 999).

OUTPUT



```
Welcome to SWI-Prolog (threaded, 64 bits,
version 8.2.4)
SWI-Prolog comes with ABSOLUTELY NO WARRAN
TY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http
s://www.swi-prolog.org
For built-in help, use ?- help(Topic). or
?- apropos(Word).

?-
% Updating index for library d:/swipl/libr
ary/
% Updating index for library d:/swipl/xpce
/prolog/lib/
% c:/Users/dell/OneDrive/Documents/Prolog/
lab2.pl compiled 0.00 sec, 8 clauses
?- location(dr_bottaci, Pl).
false.

?- ring(dr_mike, Number).
Number = 5949 .

?-  ring(Person, 5767).
Person = dr_jones .

?- ring(Person, Number).
Person = dr_jones,
Number = 5767 .

?- ring(dr_jones, 999).
false.
```



```
ring(Person, Number) :- location(Person, Place),
 phone_number(Place, Number).
location(Person, Place) :- at(Person, Place).
location(Person,Place) :- visiting(Person, Someo
ne), location(Someone, Place).
phone_number(rm303g, 5767).
phone_number(rm303a, 5949).
at(dr_jones, rm303g).
at(dr_mike, rm303a).
visiting(dr_mike, dr_jones).
```

Lab # 3

Object:

Family relationship in Prolog.

Theory:

Facts.

parent(pam,bob).

parent(tom,bob).

parent(bob,ann).

parent(bob,pat).

The parent relation has been defined by stating the n-tuples of objects based on given info in family tree. The user can easily query the Prolog system about relations defined in the program. The arguments of relations can be concrete objects or constants such as pat and ann or general objects such as X and Y. Objects of first kind are called atoms and second kind are called variables and questions to the system consists of one or more goals**.**

Additional Facts.

male(pat).

male(tom).

male(bob).

female(pam).

female(ann).

Relationship.

Mother mother(X, Y) :- parent(X,Y), female(X).

Father father(X, Y) :- parent(X,Y), male(X).

Sister sister(X.Y) :- parent(Z,X), parent(Z,Y),female(X),X\==Y.

Brother brother(X.Y) :- parent(Z,X), parent(Z,Y),male(X),X\==Y.

Has child haschild(X) :- parent(X,_).

In the above relationships _ is called the anonymous variable.

More Relationships

grandparent(X,Y) :- parent(X,Z),parent(Z,Y).

grandmother(X,Z) :- mother(X,Y),parent(Y,Z).

grandfather(X,Z) :- fatger(X,Y),parent(Y,Z).

wife(X,Y) :- parent(X,Z), parent(Y,Z),female(X),male(Y).

uncle(X,Z) :- brother(X,Y), parent(Y,Z).

Exercise:

Write a prolog program to create a family tree by creating facts and rules based on the information given below:

Parveen is the parent of Babar.

Talib is the parent of Babar.

Talib is the parent of Lubna.

Parveen is the parent of Lubna.

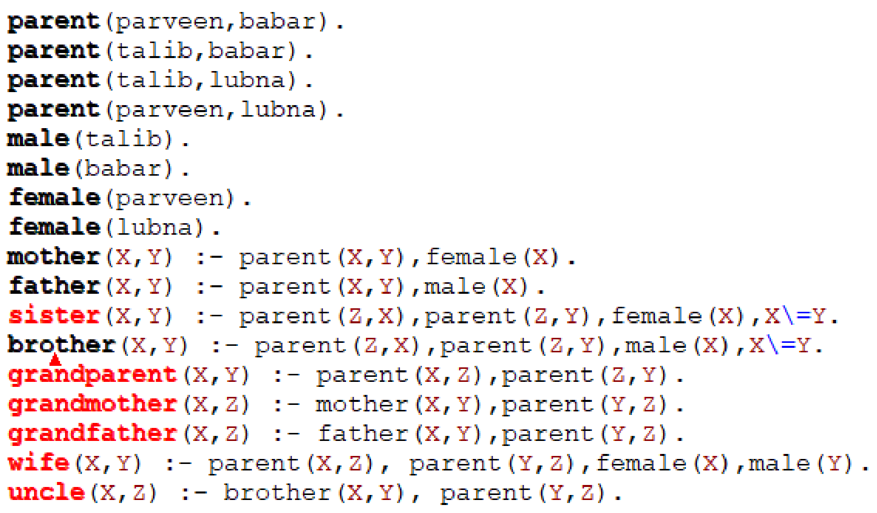
Talib is male.

Babar is male.

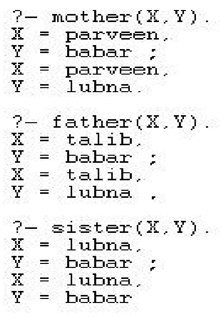
Parveen is female.

Lubna is female.

OUTPUT



```
parent(parveen,babar).
parent(talib,babar).
parent(talib,lubna).
parent(parveen,lubna).
male(talib).
male(babar).
female(parveen).
female(lubna).
mother(X,Y) :- parent(X,Y),female(X).
father(X,Y) :- parent(X,Y),male(X).
sister(X,Y) :- parent(Z,X),parent(Z,Y),female(X),X\=Y.
brother(X,Y) :- parent(Z,X),parent(Z,Y),male(X),X\=Y.
grandparent(X,Y) :- parent(X,Z),parent(Z,Y).
grandmother(X,Z) :- mother(X,Y),parent(Y,Z).
grandfather(X,Z) :- father(X,Y),parent(Y,Z).
wife(X,Y) :- parent(X,Z), parent(Y,Z),female(X),male(Y).
uncle(X,Z) :- brother(X,Y), parent(Y,Z).
```



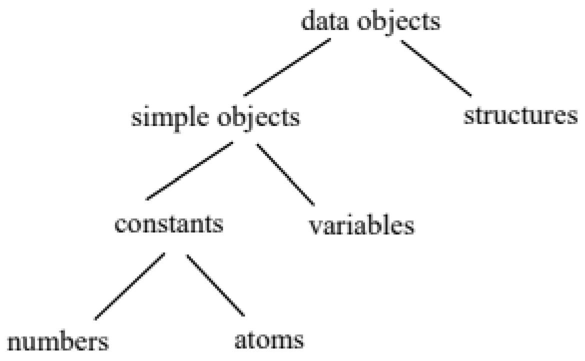
```
?- mother(X,Y).
X = parveen,
Y = babar ;
X = parveen,
Y = lubna.

?- father(X,Y).
X = talib,
Y = babar ;
X = talib,
Y = lubna .

?- sister(X,Y).
X = lubna,
Y = babar ;
X = lubna,
Y = babar
```

Lab # 4

Object:

Data Objects in Prolog.

Theory:



Examples:

Atoms tom, pat, x100, x_45.

Numbers 100, 1235, 2000.45.

Variables X, Y, X_Y, Xval.

Structures day(9, jun, 2017), point(1,4)..

Atoms.

Atoms can be constructed in three ways:

1.Strings of letters, digits and the underscore '_' starting with a lower case. Such as azhar. b59,b59_a etc.

2. Strings of special characters. Such as ->, ===, :: etc. when using atoms of this form some care is to be taken as some special character already have predefined meaning. As :-

3.Strings of characters enclosed in single quotes. This is useful when an atom needs to start with a capital letter. By enclosing it in quotes it gets distinguished from variable. Such as ‘Azhar’, ‘Prolog’

Numbers.

- Numbers can be Integers and Real.
- Integer numbers can be represented as 100   4   -87   1020
- The normal range of inter numbers is -16383 to 16383.
- The real number treatment depends on the implementation of Prolog. Examples of real numbers are 3.141 -0.00062   450.18
- Real numbers are not very much in Prolog programming as it is primarily a language for symbolic, non-numeric computation. In symbolic computation inters are often used.

Variables.

Variables are strings of letters, digits and underscore characters. They start with an upper case letter or an underscore character. Such as X, Sum, Member_name, Student_list, _a50 etc.

In a clause when a variable is used only once then the variable name can be replaced by so called anonymous variable which is written as a single underscore.

Structures.

Structured data objects or structures are objects that can have multiple components. The components can in turn be structures.

Example: the date can be viewed as structure with three components day, month and year. The date 9th June 2018 can be written as   date(9, june, 2018).

Structures can be naturally represented as trees. Prolog can be viewed as a language for processing trees.

Tree representation of structure: Prolog representation:

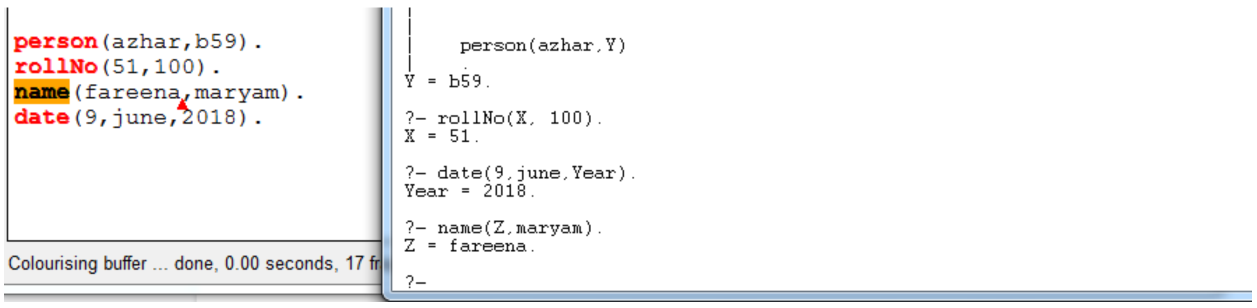
date

date(9, june, 2018)

The root of the tree is called the functor and the sub-trees are called arguments. Each functor is defined with two things: the name whose syntax is that of atom and the arity or the number of arguments.

Exercise:

Write program in prolog to implement all data types you have studied above



```
person(azhar,b59).
rollNo(51,100).
name(fareena,maryam).
date(9,june,2018).
```

Colourising buffer ... done, 0.00 seconds, 17 fr

```
|    person(azhar,Y)
|        .
Y = b59.

?- rollNo(X, 100).
X = 51.

?- date(9,june,Year).
Year = 2018.

?- name(Z,maryam).
Z = fareena.

?-
```

Lab # 5

Object:

Unifications.

Theory:

Simples unifications.
How can we ask something like "what does Fred eat?" If we have the following program:
eats(fred,oranges).

How do we ask what fred eats ? We could write something like this :

?- eats(fred,what).

But Prolog will say no. The reason is that Prolog can't find the relation eats(fred,what) in its database. In this case we have to use a variable which will be unified to match a relation given in the program. This process is known as unification. Variables are distinguished from atoms by starting with a capital letter. Here are some examples of variables:

```
X                  /* a single capital letter*/

VaRiAbLe           /* a word beginning with an upper case letter */
Two_words          /* two words separated with an underscore  */
```

Now that we know how to use a variable, we can ask the same question as before using the variable what instead of an atom.

?- eats(fred,What)
What=oranges Yes

In this case Prolog try to unified the variable with an atom. "What=oranges" means that the query is successful when what is unified with oranges. With the same program if we ask:

?- eats(Who,oranges).

In this example we ask who eats oranges. To this query Prolog should answer :
?- eats(Who,oranges).
Who=fred yes

Now if we ask:

?- eats(Who,apple).
No

Prolog answer no because he can't find in his database any atom than can match with this relation. Now if we only want to know if something is eated by anyone and we don't care about that person we can use the underscore. The '_' can be used like any variable. For example if we ask eats(fred,_) the result will be:

```
?- eats(fred,_). Yes
```

The result will be yes because Prolog can find a relation of eat between fred and something. But Prolog will not tell use the value of '_'. Now if we have the following program:

```
eats(fred,apple).
eats(fred,oranges).
```

Now if we ask:

```
?- eats(fred,What).
```

The first answer will be "What=apple" because that is the unification that match the first relation of eats with fred in the database. Then prolog will be waiting for you to press a key. If you press enter Prolog will be ready for a new query. In most implementation if you press the key ";" then Prolog will try to find if there is any other successful unification. Prolog will give the second result "What=orange" because this the second one in the program. If you press again the key ";" then Prolog will try to find a third unification. The result will be "no" because he is not able to find any other successful unification.

```
?- eats(fred,What).
What=apple;
What=oranges; no
```

Variables unification example.

Consider this example of program that could be used by a library:

```
book(1,title1,author1).
book(2,title2,author1).
book(3,title3,author2).
book(4,title4,author3).
```

Now if we want to know if we have a book from the author2 we can ask:

```
?- book(_,_,author2). yes
```

If we want to know which book from the author1 we have:

```
?- book(_,X,author1).
X=title1;
X=title2;
```

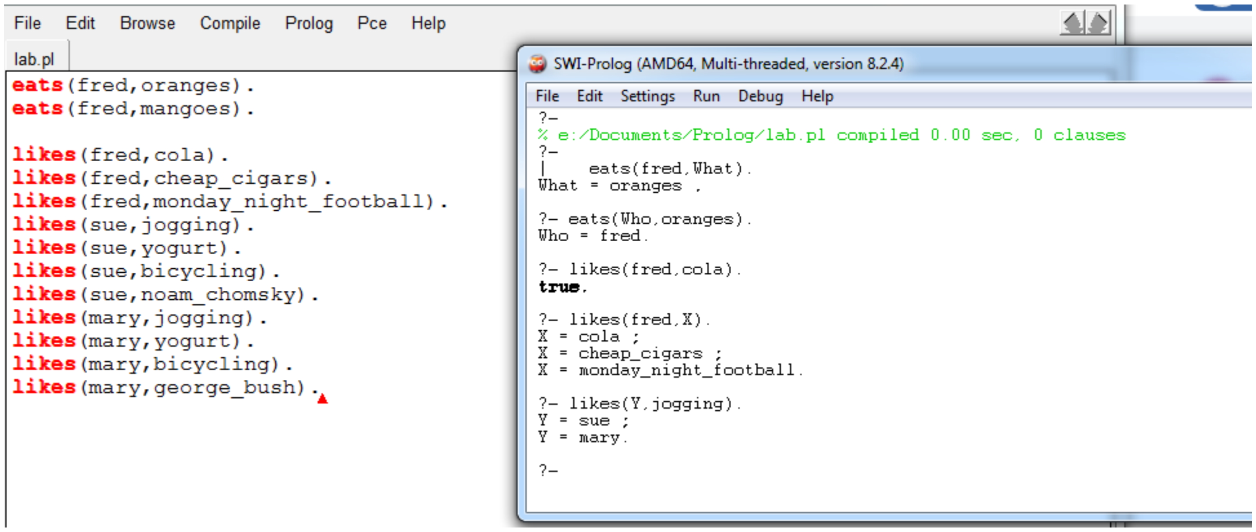
Exercise:

1. Write a program to implement the following:

```
likes(fred,cola).
likes(fred,cheap_cigars).
likes(fred,monday_night_football).
likes(sue,jogging).
likes(sue,yogurt).
likes(sue,bicycling).
likes(sue,noam_chomsky).
likes(mary,jogging).
likes(mary,yogurt).
likes(mary,bicycling).
likes(mary,george_bush).
```

Queries:

```
?- likes(fred,cola).
?- likes(fred,X).
?- likes(fred,X).
?- likes(Y,jogging).
```


File Edit Browse Compile Prolog Pce Help
lab.pl
SWI-Prolog (AMD64, Multi-threaded, version 8.2.4)
File Edit Settings Run Debug Help

```
eats(fred,oranges).
eats(fred,mangoes).

likes(fred,cola).
likes(fred,cheap_cigars).
likes(fred,monday_night_football).
likes(sue,jogging).
likes(sue,yogurt).
likes(sue,bicycling).
likes(sue,noam_chomsky).
likes(mary,jogging).
likes(mary,yogurt).
likes(mary,bicycling).
likes(mary,george_bush).
```

```
?-
% e:/Documents/Prolog/lab.pl compiled 0.00 sec, 0 clauses
?-
|    eats(fred,What).
What = oranges .

?- eats(Who,oranges).
Who = fred.

?- likes(fred,cola).
true.

?- likes(fred,X).
X = cola ;
X = cheap_cigars ;
X = monday_night_football.

?- likes(Y,jogging).
Y = sue ;
Y = mary.

?-
```

Lab # 6

Object:

Input & Output Commands.

Theory:

At this time we have seen how we can communicate with prolog using the keyboard and the screen. We will now see how we can communicate with prolog using files.

Prolog can read and write in a file. The file where Prolog read is called input and the file where Prolog write is called output. When you run Prolog the default output is your screen (the shell) and the input is your keyboard. If you want to use files for that you have to tell it to Prolog using commands.

Read and Write.

Sometimes a program will need to read a term from a file or the keyboard and write a term on the screen or in a file. In this case the goals write and read can be used.

read(X).

Read the term from the active input and unified X with it.

write(Y).

Write the term Y on the active output.

Examples.

Calculating the cube of an integer.

If we have the following program:

```
cube(C,N) :- C is N * N * N
```

If you ask something like cube(X,3) then Prolog would respond X=9. Suppose that we want to ask for other values than 3, we could write this program like this:

```
cube :-
read(X), calc(X). /* read X then query calc(X). */
calc(stop) :- !. /* if X = stop then it ends */
calc(X) :- C is X * X * X, write(C),cube. /* calculate C and write it then ask
```

again cube. */

Now if we ask Prolog cube, Prolog will ask use a value and give us the result until we write stop.

ASCII characters.

It is of course possible to use ASCII code in Prolog. For example if you type:

```
?- put(65), put(66), put(67).
```

Prolog will write ABC.

We can also write:

```
?- put(‘A’), put(‘B’), put(‘C’).
```

The output will be similar.

Tab.

The built-in predicate tab(N) causes N spaces to be output:

```
?- write(hi), tab(1), write(there),nl.
hi there
true.
?- write(hi), tab(15), write(there),nl.
hi there
true.
```

Using another program.

It is possible to load another Prolog program using a program. Two predicates have been made for this:

consult(program1) to add in the database all the predicates contained in program1

and reconsult(program2)to reload the predicates contained in program2.

Exercise:

1. Write code in prolog to generate the following output:

```
5 plus 8 is 13.
X=13
3 multiply by 3 is 9.
X = 9.
```

Program:

```
ex1:- write('5 plus 8 is 13'), nl , X is 5+8, write('X='),
tab(1),write(X),nl,nl

, write('3 multiply by 3 is 9'), A is 3*3, nl, write( 'X ='),
tab(1),write(A).
```

Output:

```
% c:/Users/User/Desktop/Lab.pl compiled 0.00 sec, 1 clauses
?- ex1.
5 plus 8 is 13
X= 13

3 multiply by 3 is 9
X = 9
true.
```

2. Write code in prolog to generate table of any number.

Program:

```
ex2:- X is 1,
    write(' enter the number '),
    read(Num),
    table(X,Num).
table(Y,N):- Y =< 10, write(N*Y),write('='), T is N*Y, write(T),
    A is Y+1,nl, table(A,N).
```

Output:

```
% c:/Users/User/Desktop/Lab.pl compiled 0.00 sec, 2 clauses

?- ex2.

 enter the number 4.

4*1=4

4*2=8

4*3=12

4*4=16

4*5=20

4*6=24

4*7=28

4*8=32
```

```
4*9=36

4*10=40

false.
```

3. Write code in prolog to print your name and roll no on screen.

Program:

```
pr :- write('Enter your name'),tab(1), read(X),write('Enter your roll
no'),read(Y),

    write( 'Name is '),tab(2), write(X),nl, write('your roll no
is'),tab(2), write(Y).
```

Output:

```
% c:/Users/User/Desktop/Lab.pl compiled 0.00 sec, 1 clauses
?- ex3.
Enter your name Mubeen Rasheed.
Enter your roll no|: 20.
Name is   Mubeen Rasheed
your roll no is 20
true.
```

LAB # 7

Object:
Operators – Arithmetic and Comparison.

Theory:
For the arithmetic operators, prolog has already a list of predefined predicates. These are: *=, is, <, >, =<, >=, ==, =:=, /, *, +, -, mod, div*

In Prolog, the calculations are not written like we have learned. They are written as a binary tree. That is to say that:
*y*5+10*x* is written in Prolog as *+(*(y,5),*(10,x)).*

But Prolog accept our way of writing calculations. Nevertheless, we have to define the rules of priority for the operators. For instance, we have to tell Prolog that * has higher priority than +..Prolog allows the programmer to define his own operators with the relations next: *Op(P, xfy, name).*

where P is the priority of the operators(between 1 and 1200), xfy indicates if the operator is infix(xfx,xfy,yfx) or postfix(fx,fy).The name is, of course, the name of the operator. Note that the prior operator has the lowest priority. Prolog has already these predefined operators: *Op(1200,xfx,':-').*
Op(1200,fx,[:-,?-]).
Op(1100,xfy,';').
Op(1000,xfy,',').
Op(700,xfx,[=,is,<,>,=<,>=,==,=:=]).
Op(500,yfx,[+.-]).
Op(500,fx,[+,-,not]).
Op(400,yfx,[,/,div]).*
Op(300,xfx,mod).

Arithmetic Operators
Prolog use the infix operator 'is' to give the result of an arithmetic operation to a variable. *X is 3 + 4.*
Prolog responds.
X = 7
yes

When the goal operator 'is' is used the variables on the right must have been unified to numbers before. Prolog is not oriented calculations, so, after a certain point, it approximates the calculations and doesn't answer the exact number:
?- X is 1000 + .0001
X = 1000
yes

Comparison Operators.
Comparison operators can be classified into Arithmetic and term comparison operators.

Less than or equal to(=<).
Compares *Arg1* to *Arg2*, succeeding if *Arg1* is less than or equal to *Arg2*. Both the arguments must be a number (integer or float) or arithmetic expression.
Arg1=< Arg2
Example:
?- 1=<2.
yes
?- 1=<1.
yes
?- 2=<1.
no
?- 1.5=<2.
yes
?- 2=<1+2.0.
Yes

Less than (<).
Compares Arg1 to Arg2, succeeding if Arg1 is less than Arg2. Both the arguments must be a number (integer or float) or arithmetic expression.
Arg1< Arg2
Example:
?- 1<2.
yes
?- 1<1.
no
?- 1.0<2.
yes
?- 2+3<6.
Yes
?- 5<3+1.
no

Greater than or equal to(>=).
Compares *Arg1* to *Arg2*, succeeding if *Arg1* is greater than or equal to *Arg2*. Both the arguments must be a number (integer or float) or arithmetic expression. *Arg1>= Arg2*

Example:
?- 2>=1.
yes
?- 2>=3.
No
?- 3+4>=7.9.
no
?- 7.5>=7+0.5.
yes

Greater than (>).
Compares Arg1 to Arg2, succeeding if Arg1 is greater than Arg2. Both the arguments must be a number (integer or float) or arithmetic expression.
Arg1> Arg2
Example:
?- 2>1.
Yes
*?- 3*3>10.*
no
?- 10>8+1.
yes
?- 11.5>10+1.
yes

Equals to (=:=).
Compares *Arg1* to *Arg2*, succeeding if *Arg1* and *Arg2* are equal. Both the arguments must be a number (integer or float) or arithmetic expression.
Arg1=:= Arg2
Example:
?- 1=:=1.
yes
?- 1=:=2.
no
?- 2+3=:=5.
yes
*?- 12=:=6*2.*
yes

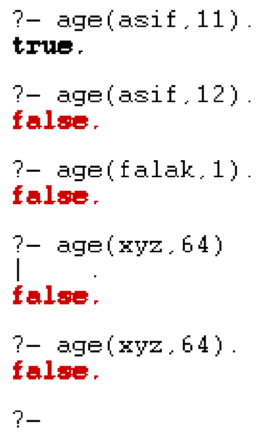
Not equals to (=\=).
Compares Arg1 to Arg2, succeeding if Arg1 and Arg2 are not equal. Both the arguments must be a number (integer or float) or arithmetic expression. *Arg1=\= Arg2* **Example:**
?- 1=\=1.
no
?- 2=\=1.
yes
*?- 3*3=\=4+5.*
no
*?- 6*3=\=10+3.*
yes
Exercise:
Write a program to compare ages by defining facts and rules from the information provided below. Use appropriate operators where necessary.
Age of aslam is 11 years.
Age of asif is 13 years.
Age of afsheen is 17 years.

Age of manal is 16 years.
Age of amir is 30 years.
Age of falak is 33 years.
Age of sobia is 40 years.
Age of sheheryar is 44 years.
Age of rehan is 52 years.
Age of erum is 64 years

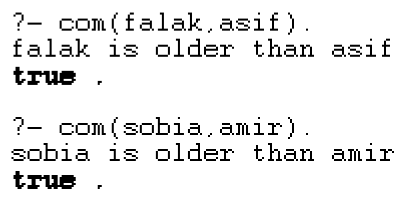
Program:

```
age(asif,11).

age(asif,30).

age(afsheen,17).

age(manal,16).

age(amir,30).

age(falak,33).

age(sobia,40).

age(shehyar,44).

age(rehan,52).

age(erum,64).

com(X,Y) :- age(X,A), age(Y,Z), A > Z, write(X),tab(1), write('is older than'),tab(1),write(Y).

com(X,Y) :- age(X,A), age(Y,Z), A < Z, write(X), write('is younger than'),write(Y).

com(X,Y) :- age(X,A), age(Y,Z), A ==Z, write(X), write('is equals to'),write(Y).
```

Output:



```
?- age(asif,11).
true.

?- age(asif,12).
false.

?- age(falak,1).
false.

?- age(xyz,64)
|    .
false.

?- age(xyz,64).
false.

?-
```



```
?- com(falak,asif).
falak is older than asif
true .

?- com(sobia,amir).
sobia is older than amir
true .
```

LAB # 8

Object:
Lists in Prolog.

Theory:
The list is a simple data structure widely used in non-numeric programming. List consists of any number of items. It can be represented in prolog as

[red, green, blue, white, dark]

A list can be either empty or non-empty. In first case the list is simply written as a Prolog atom i.e. []. In second case the list can be viewed as consisting of two things:

The first item is called the **head** of the list and the remaining part of the list is called **tail**. The tail itself has to be a list.

In the list [red, green, blue, white, dark], red is the head while rest of the elements are the tail.

List can be represented in the following manner also:
[a, b, c] = [x | [b, c]] = [a, b | [c]] = [a, b, c] | []]

List Membership.
To check whether an object X is member of list L or not.
list_member(X,L). where X is an object and L is a list.

The goal list_member(X, L) is true if X occurs in L.

Example
list_member(b, [a, b, c]) is true,
list_member(b, [a, [b, c]]) is not true,
list_member([b,c], [a,[b, c]]) is true,

The program for the membership relation can be based on the following: X is a member of L if either X is head of L or X is member of the tail of L.

```
list_member(X, [X | _ ]).
list_member(X, [_ | TAIL]) :- list_member(X,TAIL).
```

List Length Calculation.

list_length(List, N). will count the elements in a list List and instantiate N to that number. The length will be 0 if the list is empty and if the list is not empty then the length is equal to 1 plus the length of the tail

```
list_length([], 0).
list_length([_|TAIL], N) :- list_length(TAIL,N1), N is N1+1..
```

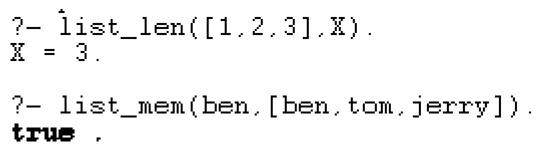
Exercise:

Write a program in Prolog and implement the member and length calculation clauses using your own data. Also write down the output.

Program:

```
list_mem(X, [X | _ ]).

list_mem(X, [_ | TAIL]) :- list_mem(X,TAIL).

list_len([], 0).

list_len([_|TAIL], N) :- list_len(TAIL,N1), N is N1+1.
```

Output:



```
?- list_len([1,2,3],X).
X = 3.

?- list_mem(ben,[ben,tom,jerry]).
true .
```

Lab # 9

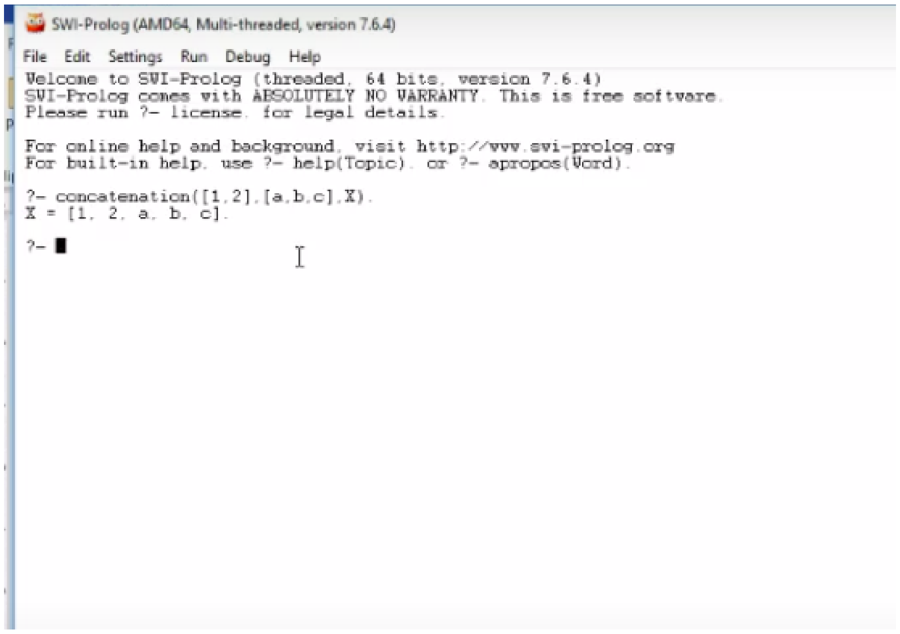
Object:
More Lists Operations

Theory:
Concatenation.
```
concatenation([], L,L).
concatenation([X1 | L1], L2, [X1 | L3] ):- concatenation(L1,L2,L3).
```
Operations in words.
```
interm(0,zero).
interm(1,one).
interm(2,two).
interm(3,three).
interm(4,four).
interm(5,five).
interm(6,six).
interm(7,seven).
interm(8,eight).
interm(9,nine).
inwords([], []).
inwords([X | TAIL], [T | Z]) :- interm(X,T), interm(TAIL, Z).
```
Delete an item.
There can be two cases; if X is the head of the list then result after deletion is tail and if X is in the tail then it is deleted from there.
```
del(Y, [Y],[]).
del(X,[X | LIST1],LIST1).
del(X,[Y | LIST], [Y | LIST1]) :- del(X,LIST,LIST1).
```
Append.
```
list_member(X, [X | _ ]).
list_member(X, [_ | TAIL]) :- list_member(X,TAIL).
list_append(A,T,T) :-list_member(A,T), !
list_append(A,TAIL,[A |TAIL])..
```
Insert
```
list_insert(X,L,R) :- list_delete(X,R,L).
list_delete(X, [X|LIST1], LIST1).
list_delete(X, [Y|LIST1], [Y|LIST1]) :- list_delete(X,LIST,LIST1)..
```
Reverse
```
list_reverse([], []).
list_reverse([First | Rest], Reversed) :- list_reverse(Rest, ReversedRest),
concatenation(ReservedRest, [First], Reserved).
concatenation([], L,L).
concatenation([X1 | L1], L2, [X1 | L3] ):- concatenation(L1,L2,L3).
```

Exercise:
Write a program in Prolog and implement the operations learned in this lab using your own data. Also write down the output.

Program

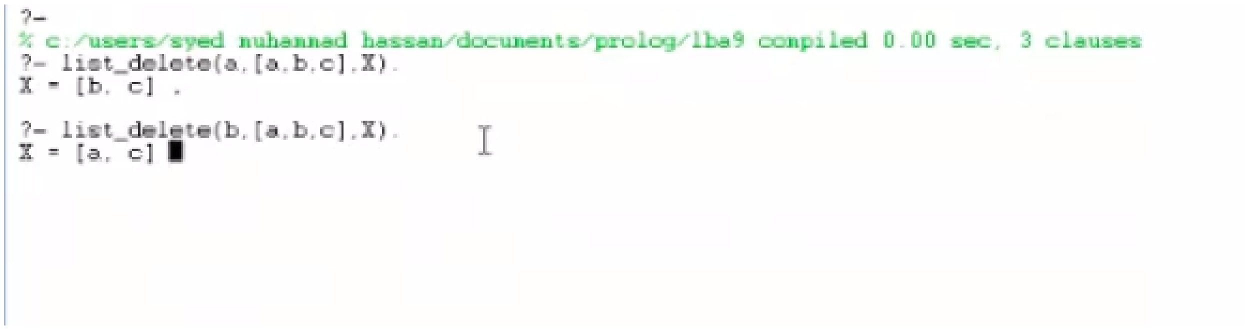
```
/*concatenation([], L,L).
concatenation([X1 | L1], L2, [X1 | L3] ):- concatenation(L1,L2,L3).
```



SWI-Prolog (AMD64, Multi-threaded, version 7.6.4)

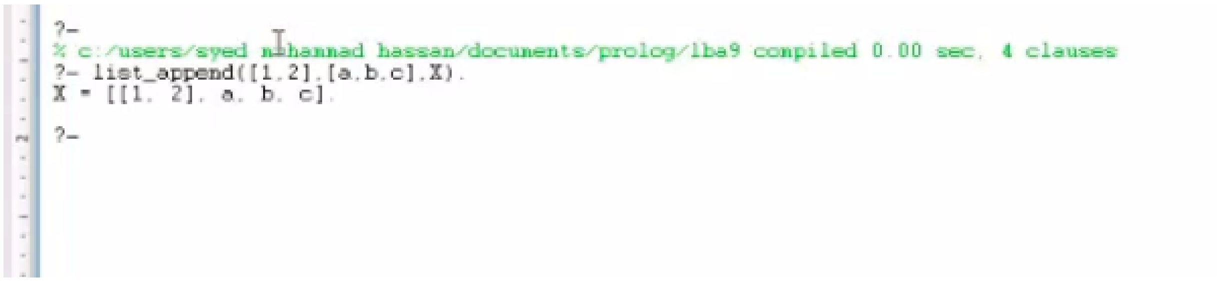
```
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- concatenation([1,2],[a,b,c],X).
X = [1, 2, a, b, c].

?- 
```

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



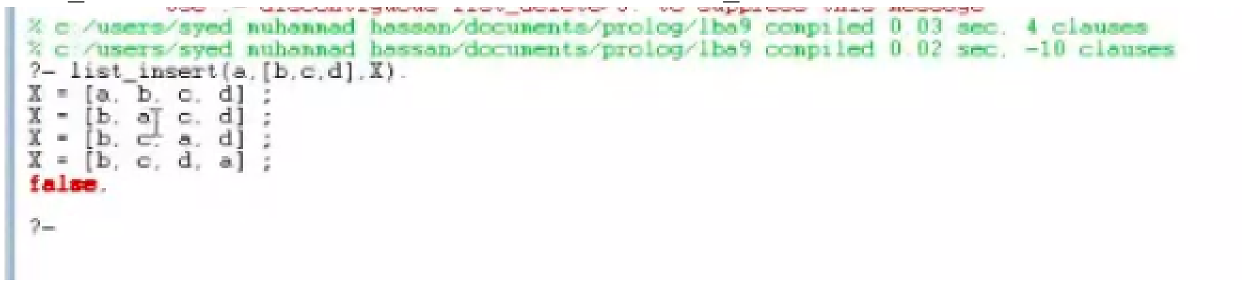
```
?-
% c:/users/syed muhammad hassan/documents/prolog/lba9 compiled 0.00 sec, 3 clauses
?- list_delete(a,[a,b,c],X).
X = [b, c] ;

?- list_delete(b,[a,b,c],X).
X = [a, c]
```

```
list_member(X, [X | _ ]).
list_member(X, [_ | TAIL]) :- list_member(X,TAIL).
list_append(A,T,T) :-list_member(A,T), !.
list_append(A,TAIL,[A |TAIL]).
```



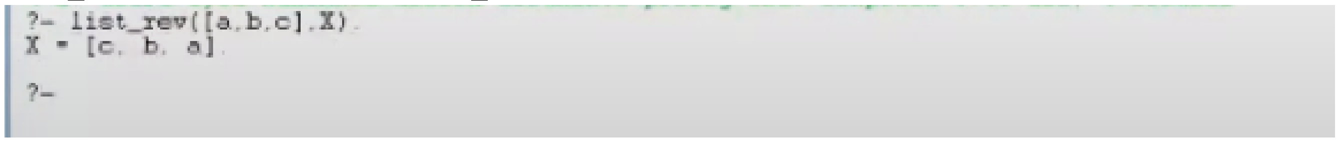
```
?-
% c:/users/syed muhammad hassan/documents/prolog/lba9 compiled 0.00 sec, 4 clauses
?- list_append([1,2],[a,b,c],X).
X = [[1, 2], a, b, c].

?-
```

```
list_insert(X,L,R) :- list_delete(X,R,L).
list_delete(X, [X|LIST1], LIST1).
list_delete(X, [Y|LIST], [Y|LIST1]) :- list_delete(X,LIST,LIST1).
```



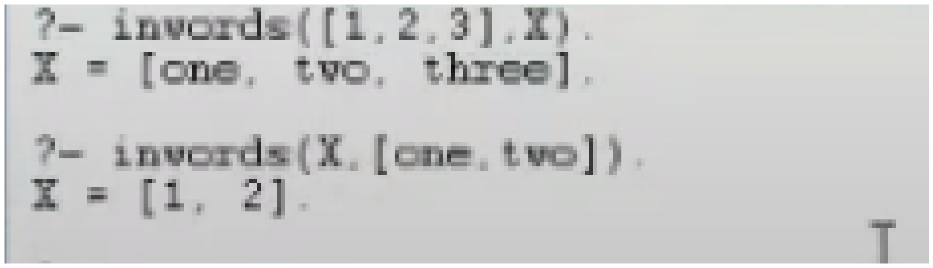
```
% c:/users/syed muhammad hassan/documents/prolog/lba9 compiled 0.03 sec, 4 clauses
% c:/users/syed muhammad hassan/documents/prolog/lba9 compiled 0.02 sec, -10 clauses
?- list_insert(a,[b,c,d],X).
X = [a, b, c, d] ;
X = [b, a, c, d] ;
X = [b, c, a, d] ;
X = [b, c, d, a] ;
false.

?-
```

```
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).
list_rev([],[]).
list_rev([Head|Tail],Reversed) :-
list_rev(Tail, RevTail),list_concat(RevTail, [Head],Reversed).
```



```
?- list_rev([a,b,c],X).
X = [c, b, a].

?-
```

```
interm(0,zero).
interm(1,one).
interm(2,two).
interm(3,three).
interm(4,four).
interm(5,five).
interm(6,six).
interm(7,seven).
interm(8,eight).
interm(9,nine).
inwords([],[]).
inwords([X|TAIL],[T|Z]) :- interm(X,T),inwords(TAIL,Z).
```



```
?- inwords([1,2,3],X).
X = [one, two, three].

?- inwords(X,[one,two]).
X = [1, 2].
```

Lab # 10

Object:
Backtracking in Prolog.

Theory:
Backtracking is a process. When a sub-goal fails, the Prolog system traces its steps backwards to the previous goal and tries to satisfy it.

This means that all variables that were bound to a value when that goal was satisfied are now made free again. Then the Prolog system tries to satisfy that goal by matching with the next clause starting at the clause just after the one that matched the sub-goal. This continues until either the sub-goal is satisfied, or until the program database has been exhausted, at which point Prolog backtracks yet again to the sub-goal that was before the current sub-goal.

Example.

```
person(alice).
person(mark).

likes(alice, coke).
likes(alice, sprite).
likes(alice, fanta).
likes(mark, pepsi).
likes(mark, coffee)

?-person(Name), likes(Name, Drink).

Name = alice.
Drink = coke.

Name = alice.
Drink = sprite.

Name = alice.
Drink = fanta.

Name = mark.
Drink = pepsi

Name = mark.
Drink = coffee.
```

Prolog automatically backtrack if it is necessary to satisfy a goal. But uncontrolled backtracking may cause inefficiency in a program. **cut (!)** can be used to prevent uncontrolled backtracking.

Example.
Find maximum of two given numbers.
max(X, Y, Max).
where max = X if X >=Y and max = Y if X <Y.

In prolog it is written as:
max(X,Y,X) :- X>=Y. max(X,Y,Y) :- X<Y.

They are mutually exclusive i.e. only one can succeed. More feasible formulation of rules using cut will be:

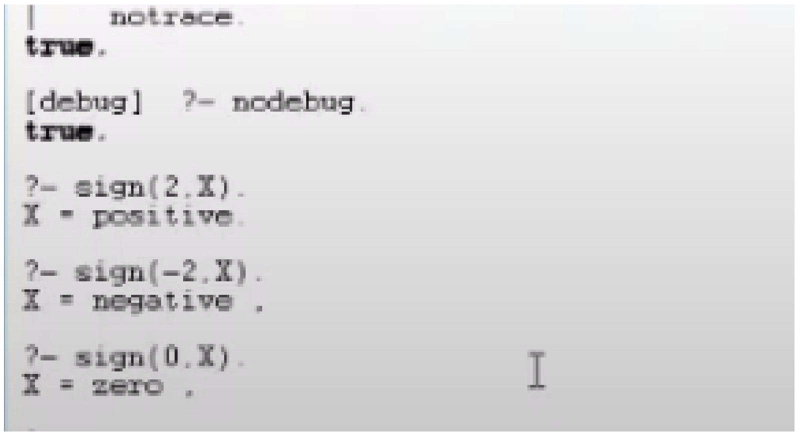
max(X, Y, Max)
where if X>-Y then Max = X
otherwise Max = Y.

In prolog:
max(X,Y,X) :- X >=Y, !.
max(X,Y,Y).
max_call(X,Y,Max) :- X>=Y, !, Max = X ; Max = Y.

Exercise:
Write a program in Prolog to find if number is positive or negative using backtracking and cut.

Program

```
sign(X, negative) :- X < 0,!.

sign(X,zero) :- X =< 0,!.

sign(X,positive) :- X > 0.
```



```
|     notrace.
true.

[debug]  ?- nodebug.
true.

?- sign(2,X).
X = positive.

?- sign(-2,X).
X = negative .

?- sign(0,X).
X = zero .
```